# 2章

# 特製ライブラリの作成方法

MSX-Cの標準ライブラリには，MSXの機能を支援するための，グラフィックスを扱う関数，あるいはジョイスティックやマウスを読み取る関数はまったく用意されていません．そこで，本書ではゲームのプログラミングに必要なこれらの関数を作成し，特製ライブラリの形にまとめて利用します．特製ライブラリは，

MSXALIB.REL…… アセンブラで作成されたライブラリ
MSXCLIB.REL…… C言語で作成されたライブラリ

の2つのファイルより成り立っています．この章では，特製ライブラリの作成方法を説明します．

# 2 1 ライブラリ作成に必要なファイル

ライブラリを作成するには，以下のファイルを用意する必要があります．巻末に付録としてこれらのリストがまとめて掲載されていますから，それぞれ指定したとおりのファイル名でMSX-DOSで動くテキストエディタを使って打ち込んでください．〝AMAKE. BAT〞〝CMAKE. BAT〞〝ACHG. BAT〞〝CCHG. BAT〞の4つのファイルは，DOSのバージョン(MSX-DOSかMSX-DOS 2か)によって異なるものを使用しますので、お手元のDOSの種類に合ったリストのほうを選んでください．そして，作成したファイルは作業用ディスクを1枚作って，そこにまとめておきましょう．

| | |
|---|---|
| MSXALIB. MAC | ……アセンブラ記述ライブラリ（MSXALIB. REL）のソース |
| AMAKE. BAT | ……MSXALIB. RELを作成するバッチプログラム |
| ACHG. BAT | ……MSXALIB. RELの一部を更新するバッチプログラム |
| MSXCLIB. C | ……MSX-C記述ライブラリ（MSXCLIB. REL）のソース |
| CMAKE. BAT | ……MSXCLIB. RELを作成するバッチプログラム |
| CCHG. BAT | ……MSXCLIB. RELの一部を更新するバッチプログラム |

特製ライブラリの作成には，以下のコマンドファイルおよび関連ファイルも必要です．これらも忘れずに用意して，作業ディスクにコピーしておいてください．ただしMSX-DOS 2をお使いの方は、〝ECHO. COM〞は必要ありません．なお〝LIB 80. COM〞と〝M 80. COM〞は『MSXDOS TOOLS』に，それ以外は『MSX-C』のシステムディスクに含まれています．

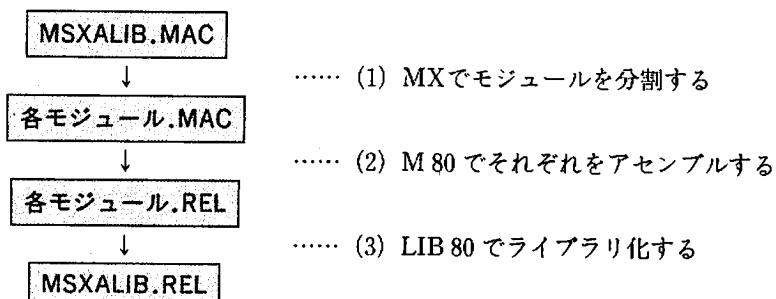
| | |
|---|---|
| LIB 80.COM | ……ライブラリ作成保守コマンド |
| M 80.COM | ……アセンブラ |
| MX.COM | ……モジュール抽出コマンド |

CF.COM　……　MSX-C パーサ
CG.COM　……　MSX-C コードジェネレータ
ECHO.COM　……　ファイルに文字列を追加する（MSX-DOS 2 では必要なし）
AREL.BAT　……　MX.COM が使うデータファイル
CREL.BAT　……　MX.COM が使うデータファイル
STDIO.H　……　MSX-C のインクルードファイル

## 特製ライブラリの作成手順

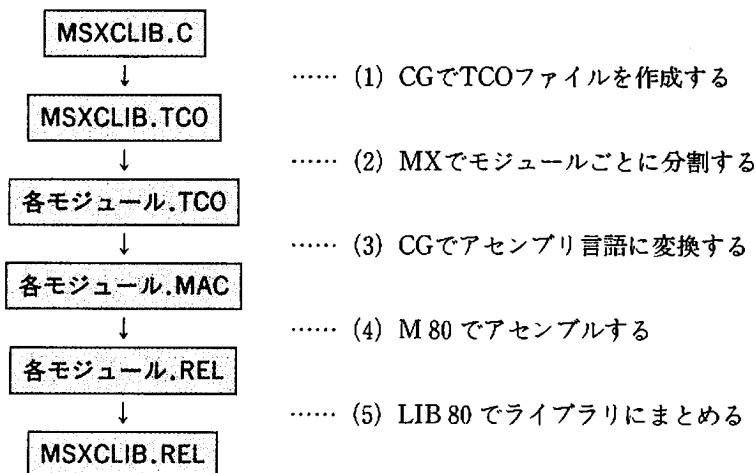
本書で作成する特製ライブラリは，アセンブラで記述した部分と，MSX-C で記述した部分の 2 つにわかれています．

### ● 〝MSXALIB.REL〟の作成手順

まず最初に，アセンブラ記述ライブラリ〝MSXALIB.REL〟の作成方法を説明しましょう．ここには，BIOSコール関数や，特に実行スピードの速さが要求される関数が含まれています．ソースプログラム〝MSXALIB.MAC〟からライブラリ〝MSXALIB.REL〟を完成させるには次の手順が必要です．

以上をオートマチックに実行するバッチファイルが〝AMAKE.BAT〟です．アセンブラのソース〝MSXALIB.MAC〟を入力して，このバッチファイルを実行すれば，次のようにすべての処理がおこなわれ，最終的に〝MSXALIB.REL〟が完成します．

```
A> amake
```

● “MSXCLIB.REL” の作成手順

もうひとつのライブラリ “MSXCLIB. REL” は，MSX-Cで記述されています．ソースプログラム“MSXCLIB. C”からライブラリ “MSXCLIB. REL” を完成させるには次の手順が必要です．

以上をオートマチックに実行するバッチファイルが “CMAKE. BAT” です．MSX-Cのソース “MSXCLIB. C” および “MSXȦLIB. H”，MSX-Cの標準インクルードファイル “STDIO. H” を用意して，このバッチファイルを実行すれば，次のようにすべての処理がおこなわれ，最終的に “MSXCLIB. REL” が完成します．

```
A> cmake
```

## 特製ライブラリの利用前の準備

特製ライブラリを使っているプログラムをコンパイルする場合には，コンパイル用バッチプログラム〝CC.BAT〟をリスト2.1のように変更してください．

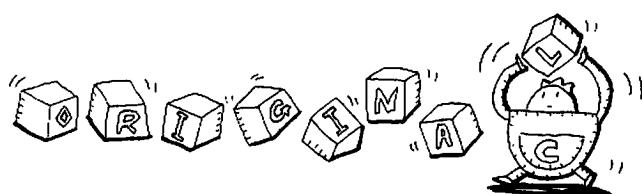
また，2つのライブラリファイルそれぞれに対して，ライブラリに含まれる関数の宣言をおこなうヘッダーファイルが必要です．巻末に

MSXALIB.H および
MSXCLIB.H

として掲載してありますから、これらを入力し〝MSXALIB.H〟および〝MSXCLIB.H〟というファイル名をつけて保存してください．

以上で特製ライブラリを利用するためのすべての作業の完了です．

```
cf %1
cg %1
m8Ø =%1/z
del %1.tco
l8Ø ck,%1,msxclib/s,msxalib/s,clib/s,crun/s,cend, %1/n/e
del %1.mac
del %1.rel
```

リスト2.1 CC.BAT

# 2-2 ライブラリの保守

〝MSXALIB. REL〞や〝MSXCLIB. REL〞の作成は，最初から最後まで通しておこなうと，たいへん時間がかかります．ですから，ソースプログラムを更新するたびにすべてをやり直すのはひどく手間な作業です．そこで，1つの関数のソースプログラムを更新したならば，その関数だけについて再コンパイル，あるいは再アセンブルして，現存するライブラリと差し替えるためのバッチプログラムを用意しました．ソースプログラムの一部分だけ訂正するような場合に利用してください．

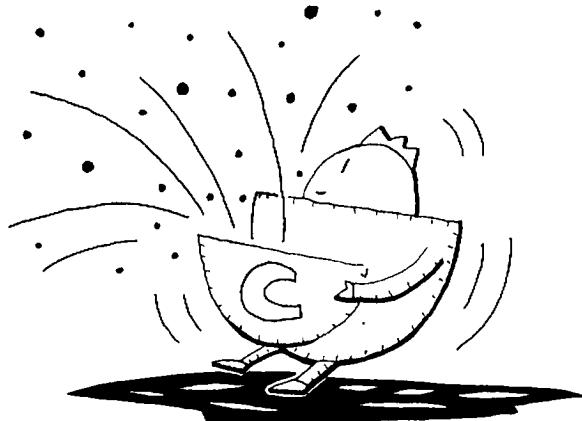
〝MSXALIB. REL〞を更新する場合には〝ACHG. BAT〞を使用します．たとえばline関数のモジュールを変更するなら次のように入力します．これで〝MSXALIB. REL〞の中のline関数を更新することができます．

```
A> achg line    …… MSXALIB. RELの中のline関数を更新する
```

〝MSXCLIB. REL〞の更新には〝CCHG. BAT〞を使用します．たとえばscreen関数のモジュールを変更するなら次のように入力します．これで〝MSXCLIB. REL〞の中のscreen関数を更新することができます．

```
A> cchg screen …… MSXCLIB. RELの中のscreen関数を更新する
```

# 3章

# 特製ライブラリ関数の説明

特製ライブラリの入力から作成まで，うまく終了したでしょうか．ごくろうさまです．あれま，コンパイルエラーばかりでる？　そういう人は，あまり根をつめずに時間をおいて，再び入力したソースプログラムをチェックしてください．きっと，どこかに入力ミスがあるはずですから．

さあ，ここからは実際に特製ライブラリの使い方を見ていきましょう．本章は，28個あるライブラリの完全リファレンスマニュアルになっています．各関数の使い方はもとより，多くのものにはサンプルプログラムが掲載されています．実際に入力して，いろいろ試してみてください．

# 特製ライブラリの全関数

特製ライブラリには次に示したように，いくつかのカテゴリーに分類できる全28種類の関数が用意されています．各関数の解説は，この章で1つずつ懇切丁寧にまとめてあります．サンプルプログラムなどを併用して，十分使い方を覚えてください．

・ゲームで使われる関数

rnd() …… 乱数を生成する
srnd() …… 乱数系列を変更する
sgn() …… 数値の符号を調べる

・グラフィックに関係するもの

ginit() …… グラフィックを使用する前の初期設定
screen() …… 画面のモードを設定する
color() …… 画面のカラーを設定する
palett() …… パレットを変更する
line() …… グラフィック画面に直線を引く
boxfil() …… グラフィック画面の長方形を塗りつぶす
cpym2v() …… グラフィック画面にパターンを表示する
cpyv2v() …… グラフィック画面のパターンを移動させる

・スプライトに関係するもの

inispr() …… スプライトを初期化しサイズを決定する
sprite() …… スプライトパターンを定義する
putspr() …… スプライトを表示する
colspr() …… スプライトの1ラインごとの色を指定する

・VRAMに関係するもの

vpoke() …… VRAMへ1バイトのデータを書き込む
vpeek() …… VRAMから1バイトのデータを読み出す
ldirvm() …… VRAMへ一度に大量のデータを書き込む
ldirmv() …… VRAMから一度に大量のデータを読み出す
filvrm() …… VRAMの連続した範囲を一定のデータで埋める

・入力に関係するもの

snsmat()　……　押されたキーを即座に読み取る
kilbuf()　……　キーの先読みバッファをクリアする
gtstck()　……　ジョイスティックの状態を読み取る
gttrig()　……　トリガーボタンの状態を読み取る
gtpad()　……　マウスの状態を読み取る

・PSGに関係するもの

gicini()　……　PSGを初期化する
sound()　……　いろいろな音程の音を鳴らす

・BIOSに関係するもの

calbio()　……　ROM-BIOSを呼び出す

## 凡例

この章では以下の形式に従ってライブラリ関数の説明をおこないます.

# rnd関数

【機能】 乱数を生成します

【仕様】 #include <msxalib. h>

```
unsigned rnd (n)
unsigned n;        …… 1から65535までの整数
```

【説明】

0からn-1までの範囲でランダムな整数を返します．乱数はゲームには不可欠なものですが，MSX-CにはBASICのように手軽に利用できる乱数がありません．そこで用意したのがこのrnd関数です．これはある数式に従って整数を順次発生させているもので，完全な乱数とはいえません．しかし実用上はこれで十分です．

【用例】

10個の乱数を表示します．

乱数とはいっても，実行するごとに同じ数列が表示されることに気付かれたと思います．なぜなら，このrnd関数は，ある数式により次々と数を計算していくのですが，その乱数の種（いちばん最初の値）に必ず同じものを使うからです．これでは何度ゲームをしても毎回まったく同じ進行ということにもなりかねません．そこで利用するのが，次で紹介するsrndという関数です．

```
Ø: #include <stdio.h>
1: #include <msxalib.h>
2:
3: VOID main()
4: {
5:     int i, r;
6:
7:     for (i = Ø; i < 1Ø; ++i) {
8:         r = rnd(1ØØØ);
9:         printf("%3d ", r);
1Ø:     }
11: }
```

リスト3.1　RND.C

# srnd関数

【機能】 乱数系列を変更します

【仕様】 #include 〈msxalib. h〉

VOID srnd ( )

【説明】

この関数は発生する乱数の並びを変化させます. 1/60秒ごとにカウントアップするハードウェアカウンタを乱数の種に利用するため, 関数の実行後どのような乱数系列が得られるかは, まったく予測がつきません. ゲームの敵の動きを決める場合などには格好のものでしょう.

【用例】

rnd関数のサンプルプログラムに, srnd関数を付け足したものです.

実行すると, 毎回異なる10個の乱数を表示します. 実行するごとに乱数列が変化するところに注目してください.

```
 0: #include <stdio.h>
 1: #include <msxalib.h>
 2: #include <msxclib.h>
 3:
 4: VOID main()
 5: {
 6:     int i, r;
 7:
 8:     srnd();    ............................... このsrnd関数に注目
 9:     for (i = 0; i < 10; ++i) {
10:         r = rnd(1000);
11:         printf("%3d ", r);
12:     }
13: }
```

リスト3.2 SRND.C

# sgn関数

【機能】 数値の符号を調べます

【仕様】 #include　<msxclib.h>

```
int sgn (n)
int n;      …… -32768～+32767 の整数
```

【説明】

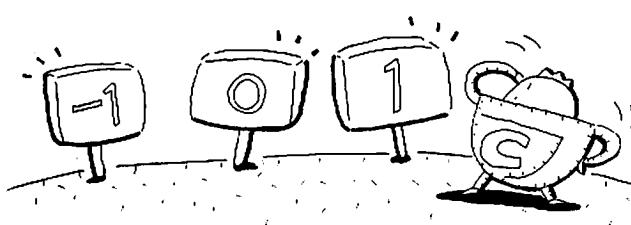
sgn関数は引き渡されたパラメータの数値が正の数のときは 1, 0 のときは 0, 負の数のときは-1 の値を返します. この関数は移動方向や対象物の存在するベクトル方向などを知るのに利用されます.

【用例】

ランダムな位置に現れた文字が画面の中央に向かって動いていきます.

最初に定義しているplot関数は, 座標 (x, y) に文字chを表示させるためのものです. main関数のfor文では, まず乱数で最初の位置 (x, y) が決まります. このあと, x座標については,「x ＋＝ sgn(20-x)」という式で次の位置を計算します. これはsgn関数の働きによって,「xが 20 より小さければ 1 を加え, xが 20 より大きければ-1 を加え, xが 20 に等しければ 0 を加える」という意味になりますから, この計算を繰り返せば, xの値がなんであっても最終的には 20 に収束するわけです. 同様にy座標も「y ＋＝ sgn (10-y)」という計算を繰り返すことで, 最終的に 10 に向かって収束していきます.

```
0: #include <stdio.h>
1: #include <msxalib.h>
2: #include <msxclib.h>
3:
4: VOID plot(x, y, ch)    ……. 座標(x,y)に文字 ch を表示する
5: int x, y, ch;
6: {
7:     printf("¥33Y%c%c%c¥n", y+32, x+32, ch);
8: }
9:
10: VOID main()
11: {
12:     int x, y, c;
13:
14:     srnd();
15:     puts("¥33x5¥f");    ………….. カーソル消去，画面クリア
16:
17:     for (c = 'A'; c <= 'Z'; ++c) {
18:         x = rnd(40);
19:         y = rnd(22);
20:         do {
21:             plot(x, y, ' ');
22:             x += sgn(20 - x);    ………. 次の x 座標を計算
23:             y += sgn(10 - y);    ………. 次の y 座標を計算
24:             plot(x, y, c);
25:         } while (x != 20 || y != 10);
26:     }
27: }
```

リスト3.3 SGN.C

# ginit関数

**【機能】** グラフィックを使用する前の初期設定

**【仕様】** #include　<msxalib.h>

```
VOID ginit ( )
```

**【説明】**

ginit関数は，VDPのポートアドレスを調べる関数です．この後で述べるいくつかのグラフィック処理関数は，I/Oポートを通してVDPを操作するため，それらの関数を使用する前にはginit関数を必ず呼び出さなければなりません．

ポートとはCPUが外部の装置とデータをやりとりするために必要な窓口と思ってください．VDPの他にも，プリンタやフロッピーディスクなど，さまざまな装置がそれぞれのポートを通してCPUとつながっています．

ほとんどのMSXでは，VDPのポートは決まった位置（ポートアドレス）にありますが，ごく一部の機種，あるいはバージョンアップアダプタを使用したときは異なる可能性があります．そのような場合に備えて，VDPのポートアドレスを調べるのが，ginit関数です．

より信頼性の高いプログラムにするためにも，このポートアドレスは必ず調べてください．

**【用例】**

次の各関数の項を参照してください．

| | | | | |
|---|---|---|---|---|
| palett() | line() | boxfil() | cpym2v() | cpyv2v() |
| vpoke() | vpeek() | ldirvm() | ldirmv() | filvrm() |

# screen関数

【機能】 画面のモードを設定します

【仕様】 #include <msxclib.h>

```
VOID screen (md)
int md;      …… 画面モード番号
```

【説明】

使用する画面モードを指定します．この関数は，MSX BIOSの005Fhをコールすることで実現しています．

SCREEN 0～3はすべてのMSX機種で，SCREEN 4～8はMSX 2/2+で，SCREEN 10～12はMSX 2+以降で使用できます．パラメータmdが示す画面モードのそれぞれの機能は表3.1のとおりです．

| 画面モード | スクリーン形式 | ドットまたは文字数 | 表示色 |
|---|---|---|---|
| SCREEN 0 | テキスト | 80 (40)×24 文字 | 512 色中 2 色 |
| SCREEN 1 | テキスト | 32×24 文字 | 512 色中 16 色 |
| SCREEN 2 | グラフィック | 256×192 ドット | 512 色中 16 色 |
| SCREEN 3 | グラフィック | 64×48 ドット | 512 色中 16 色 |
| SCREEN 4 | グラフィック | 256×192 ドット | 512 色中 16 色 |
| SCREEN 5 | グラフィック | 256×212 ドット | 512 色中 16 色 |
| SCREEN 6 | グラフィック | 512×212 ドット | 512 色中 4 色 |
| SCREEN 7 | グラフィック | 512×212 ドット | 512 色中 16 色 |
| SCREEN 8 | グラフィック | 256×212 ドット | 256 色 |
| SCREEN 10 | グラフィック | 256×212 ドット | 12,499 色 |
| SCREEN 11 | グラフィック | 256×212 ドット | 12,499 色 |
| SCREEN 12 | グラフィック | 256×212 ドット | 19,268 色 |

注：グラフィックの画面モードを使用したら，最後に必ずテキストの画面モードにもどしてください．

表3.1 画面モード一覧

もっとも，こんなに多くの画面モードがあっても，使いやすいモードは限られています．ゲームのプログラミングにはSCREEN 1とSCREEN 5がおすすめでしょう．

SCREEN 1は文字表示だけのモードです．文字だけとはいっても，フォントは自由に変えられますし，文字に色を付けることもできますから，工夫次第でけっこう見栄えのする画面が作れます．

グラフィックス表示をおこないたい場合には，ある程度きれいな絵が表示でき，かつ表示も高速なSCREEN 5がよいでしょう．このモードはプログラミングのしやすさの面からも，MSX-Cによるゲームプログラミングには最適の画面モードではないかと思います．本書の後半では，この画モードの特徴をそれぞれ生かしたプログラムを作ってみることにします．

【用例】

まずSCREEN 1の画面に「SCREEN 1」と表示し，何かキーを押すとSCREEN 0にモードを切り替えて「SCREEN 0」と表示します．

ここではテキスト画面(SCREEN 0とSCREEN 1)しか指定しませんでしたが，もちろんscreen関数はグラフィック画面も指定できます．ただしその場合に気を付けてほしいのは，プログラムの最後で必ずテキストモードに戻しておくことです．そうしないとグラフィック画面のままMSX-DOSに戻ってしまい，キー入力が画面に表示されなくなってしまいます．BASICと違い，MSX-Cにはプログラムが終了しても自動的にテキスト画面に戻すような親切な機能はありません．

```
 0: #include <stdio.h>
 1: #include <msxclib.h>
 2:
 3: VOID main()
 4: {
 5:     screen(1);   ……………………… SCREEN 1を指定
 6:     puts("SCREEN 1");
 7:     getch();     ……………………… キーが押されるのを待つ
 8:     screen(0);   ……………………… SCREEN 0を指定
 9:     puts("SCREEN 0");
10: }
```

リスト3.4　SCREEN.C

# color関数

【機能】 画面のカラーを設定します

【仕様】 #include <msxclib.h>

```
VOID color (fg, bg, bd)
int fg;        …… 前景色
int bg;        …… 背景色
int bd;        …… 周辺色
```

**【説明】**

画面の文字，背景，周辺などを指定するカラーに変えます．この関数は，MSX BIOSの0062hをコールすることで実現しています．

前景色とは表示する文字の色のことで，背景色とは画面の地の色，また周辺色とは画面の周辺部，言い換えれば額縁のような部分の色のことです．

色の指定はパレット番号または色番号によっておこないます．指定できる値の範囲は表3.2のとおりです．グラフィックの画面においては前景色を指定しても無意味ですが，関数としては値を渡す義務があるので仮に数値の0を与えてください．

**【用例】**

背景色と周辺色を，カラー番号1から15までのすべての組み合わせについて表示していきます．

プログラムを実行するとわかりますが，周辺色と背景色はcolor関数を実行した瞬間に変化します．しかし，テキスト画面に文字を表示する場合，前景色を変えただけでは文字の色は変化しません．文字の色を変えるには，color関数の後でscreen関数を一度実行しなくてはなりません．このことを忘れないよう注意してください．

| 画面モード | 指定色 | 指定範囲 | 前景色 | 背景色 | 周辺色 |
|---|---|---|---|---|---|
| SCREEN 0 | パレット | 0~15 | ○ | ○ | ○ |
| SCREEN 1 | パレット | 0~15 | ○ | ○ | ○ |
| SCREEN 2 | パレット | 0~15 | × | △ | ○ |
| SCREEN 3 | パレット | 0~15 | × | △ | ○ |
| SCREEN 4 | パレット | 0~15 | × | △ | ○ |
| SCREEN 5 | パレット | 0~15 | × | △ | ○ |
| SCREEN 6 | パレット | 0~3 | × | △ | ○ |
| SCREEN 7 | パレット | 0~15 | × | △ | ○ |
| SCREEN 8 | 色番号 | 0~255 | × | △ | ○ |
| SCREEN 10 | パレット | 0~15 | × | △ | ○ |
| SCREEN 11 | 色番号 | 0~255 | × | △ | ○ |
| SCREEN 12 | 色番号 | 0~255 | × | △ | ○ |

注：×は指定しても無意味ですが，通常は0を設定しておいてください．
△はscreen関数により画面のモードを設定したときに反映されます．

表3.2 color関数で指定できる色番号

```
 0: #include <stdio.h>
 1: #include <msxclib.h>
 2:
 3: VOID main()
 4: {
 5:     int  i, j, k;
 6:
 7:     screen(1);
 8:
 9:     for (i = 1; i <= 15; ++i) {
10:         for (j = 1; j <= 15; ++j) {
11:             color(15, j, i);   ………… 背景色 j, 周辺色 i
12:             for (k = 0; k < 18000; ++k);
13:         }
14:     }
15:
16:     color(15, 1, 1);   ………… カラーを元に戻す
17:     screen(0);
18: }
```

リスト3.5 COLOR.C

# palett関数

【機能】 指定したパレットを変更します (ginit() が必要)

【仕様】 #include <msxalib.h>

```
VOID palett (pl, r, g, b)
int pl;        …… パレット番号
int r;         …… 赤の輝度
int g;         …… 緑の輝度
int b;         …… 青の輝度
```

【説明】

画面の色は光の3原色である赤, 緑, 青の明るさの組み合わせによってさまざまに変化します. パレット機能は, この原色それぞれの明るさ (輝度) を0～7の8段階に指定するものです. 0はその原色が全く無い状態, 7は一番明るい状態を表しています. パレットをいろいろと変化させると, 画面に描かれた絵を点滅させたり, 瞬時に色を変えてしまうこともできるのです.

パレット機能はMSX 2/2+のSCREEN 0～7, 10で使用できます. color関数の項で述べたとおり, SCREEN 6では0～3, それ以外では0～15までのパレット番号が指定できます. 表3.3にカラーパレットの電源投入時の初期状態を示します.

【用例】

パレット操作により画面の背景の色を周期的に変化させます.

まず背景色をパレットコード1(通常は黒)に設定します. そして, このパレットに配列Red [14], Grn [14], Blu [14] に用意された赤, 緑, 青の輝度を繰り返し順番に割り当てていきます. 何かキーがおされたら, パレットコード1の色を黒に戻して終了します.

| パレット | 色 | 赤 | 青 | 緑 | パレット | 色 | 赤 | 青 | 緑 |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 透明 | 0 | 0 | 0 | 8 | 赤 | 7 | 1 | 1 |
| 1 | 黒 | 0 | 0 | 0 | 9 | 明るい赤 | 7 | 3 | 3 |
| 2 | 緑 | 1 | 1 | 6 | 10 | 暗い黄 | 6 | 1 | 6 |
| 3 | 明るい緑 | 3 | 3 | 7 | 11 | 明るい黄 | 6 | 3 | 6 |
| 4 | 暗い青 | 1 | 7 | 1 | 12 | 暗い緑 | 1 | 1 | 4 |
| 5 | 明るい青 | 2 | 7 | 3 | 13 | マゼンタ | 6 | 5 | 2 |
| 6 | 暗い赤 | 5 | 1 | 1 | 14 | 灰 | 5 | 5 | 5 |
| 7 | シアン | 2 | 7 | 6 | 15 | 白 | 7 | 7 | 7 |

表3.3　カラーパレットの初期化色とパレット設定値

同じような動作を，暗い赤と明るい赤の間で繰り返すと，シューティングゲームによくみられる火口などの熱源体の点滅状態になります．

```
Ø: #include <stdio.h>
1: #include <msxalib.h>
2: #include <msxclib.h>
3:
4: int Red[14] = {1,2,3,4,5,6,7,6,5,4,3,2,1,Ø};
5: int Grn[14] = {1,Ø,1,2,3,4,5,6,7,6,5,4,3,2};
6: int Blu[14] = {3,2,1,Ø,1,2,3,4,5,6,7,6,5,4};
7:
8: VOID main()
9: {
1Ø:     int i, j, k;
11:
12:     ginit();   ……………………… palett実行時にはginitが必要
13:     color(15, 1, 1);
14:
15:     while (!kbhit()) {   ………………… なにかキーを押すまで続ける
16:         for (i = Ø; i < 14; ++i) {
17:             palett(1, Red[i], Grn[i], Blu[i]);   ……… パレットの変更
18:             for (k = Ø; k < 1ØØØ; ++k);
19:         }
2Ø:     }
21:
22:     palett(1, Ø, Ø, Ø);   ……………………… 最後にパレットを戻しておく
23: }
```

リスト3.6　PALETT.C

# line関数

**【機能】** グラフィック画面上に直線を引きます（ginit() が必要）

**【仕様】** #include <msxalib.h>

```
VOID line (x1, y1, x2, y2, cl, lg)
int x1; ······ x座標の始点
int y1; ······ y座標の始点
int x2; ······ x座標の終点
int y2; ······ y座標の終点
int cl; ······ パレット番号あるいはカラー番号
int lg; ······ ロジカルオペレーション
```

**【説明】**

このline関数はSCREEN 5～12専用（本関数はVDPを直接アクセスすることにより実現しています．したがってSCREEN 2～4では使用できません）で，グラフィック画面上の座標（x1,y1）と座標（x2,y2）を第5パラメータclが示す色の直線で結びます．なお第6パラメータlgは表3.4のようなロジカルオペレーションです．各ロジカルオペレーションの機能については，cpym2v関数の項をあわせて参照してください．

| 値 | 演算 | 意味 |
|---|---|---|
| 0 | PSET | 指定色で描画 |
| 1 | AND | 指定色と背景色のANDをとった色で描画 |
| 2 | OR | 指定色と背景色のORをとった色で描画 |
| 3 | XOR | 指定色と背景色のXORをとった色で描画 |
| 4 | PRESET | 指定色を反転した色で描画 |

表3.4 ロジカルオペレーション

【用例】

直線だけでカレイドスコープ（万華鏡）を描きます．

座標（x 1, y 1）は画面を斜めに移動していきます．座標（x 2, y 2）は画面の中心に関してそれと点対称な位置です．プログラムは，何かキーが押されるまでこの両点を対角線とする4角形を描き続けます．直線の色は0～15までを繰り返し，描画のロジカルオペレーションに3（XOR）を使っているため，画面上では色が干渉しあってきれいな模様になります．

```
 0: #include <stdio.h>
 1: #include <msxalib.h>
 2: #include <msxclib.h>
 3:
 4: VOID main()
 5: {
 6:     unsigned x1, y1, x2, y2, c, i, j;
 7:
 8:     ginit();    ...................... line 実行時には ginit が必要
 9:     screen(5);
10:     color(15, 1, 1);
11:     x1 = 128;
12:     y1 = 106;
13:     c = 0;
14:     i = j = 1;
15:
16:     while (!kbhit()) {    ......... キーが押されるまで繰り返す
17:         x2 = 256 - x1;
18:         y2 = 212 - y1;
19:         line(x1, y1, x2, y1, c, 3);
20:         line(x1, y2, x2, y2, c, 3);
21:         line(x1, y1, x1, y2, c, 3);
22:         line(x2, y1, x2, y2, c, 3);
23:         if ((x1 += i) == 6 || x1 == 250) i = -i;
24:         if ((y1 += j) == 2 || y1 == 210) j = -j;
25:         if (++c == 16) c = 0;
26:     }
27:
28:     screen(0);
29: }
```

リスト3.7　LINE.C

# boxfil関数

**【機能】** グラフィック画面上の四角形を塗りつぶします（ginit() が必要）

**【仕様】**
```
#include <msxalib.h>
VOID boxfil (x1, y1, x2, y2, cl, lg);
int x1; …… x座標の始点
int y1; …… y座標の始点
int x2; …… x座標の終点
int y2; …… y座標の終点
int cl; ………… パレット番号あるいはカラー番号
int lg; ………… ロジカルオペレーション
```

**【説明】**

このboxfil関数はSCREEN 5～12 専用（本関数はVDPを直接アクセスすることにより実現しています．したがってSCREEN 2～4 では使用できません）で，グラフィック画面上の座標（x 1, y 1）と座標（x 2, y 2）を対角とする四角形を第5パラメータclが示す色で塗りつぶします．第6パラメータlgはロジカルオペレーションです．ごく普通に四角形を指定色で塗るだけなら，lgには0を指定してください．各ロジカルオペレーションの機能については，cpym 2 v関数の項をあわせて参照してください．

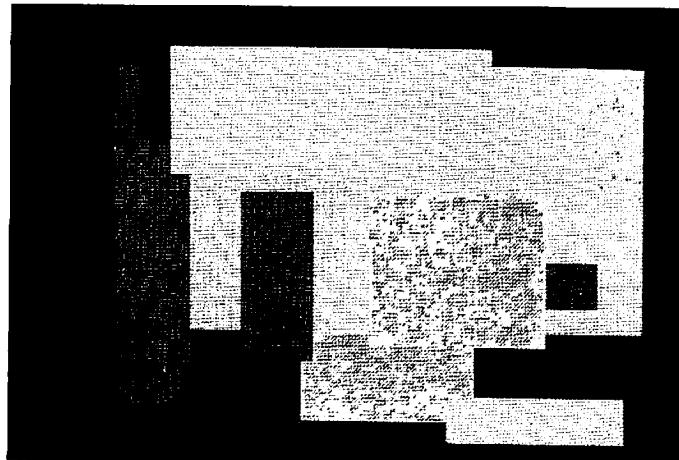
**【用例】**

四角形をランダムに描きます．

四角形をSCREEN 8の画面上のランダムな位置に，ランダムな大きさで，そしてランダムな色で描きます．この動作は何かキーを押すまで続けられます．

```
0: #include <stdio.h>
 1: #include <msxalib.h>
 2: #include <msxclib.h>
 3:
 4: VOID main()
 5: {
 6:     int i;
 7:
 8:     ginit();    ...................... boxfil実行時にはginitが必要
 9:     srnd();
10:     screen(8);
11:
12:     while (!kbhit()) {
13:         boxfil(rnd(256), rnd(212),
14:                rnd(256), rnd(212), rnd(256), 0);
15:         for (i = 0; i < 8000; ++i);
16:     }
17:     screen(0);
18: }
```

リスト3.8　BOXFIL.C

重なり合う四角形の群

BOXFIL.Cの実行画面

# cpym2v関数

**【機能】** グラフィックパターンを画面上に表示します（ginit() が必要)

**【仕様】** #include <msxalib.h>

```
VOID cpym 2 v (sr, dr, dx, dy, dp, lg)
char *sr; ………… RAM上の画像データへのポインタ
int dr; ………… 転送方向
int dx; …… 転送先のx座標
int dy; …… 転送先のy座標
int dp; ………… 転送先のVRAMのページ番号
int lg; ………… ロジカルオペレーション
```

**【説明】**

このcpym2v関数はSCREEN 5～12 専用（本関数はVDPを直接アクセスすることにより実現しています．したがってSCREEN 2～4 では使用できません）で，第1パラメータsrが示す画像データを，第5パラメータdpが示すページの，座標(dx, dy)の位置に表示します．第2パラメータdrで転送方向を指定することにより，図3.1のようにデータの上下を反転させたり左右を反転させることもできます．指定できるページ番号および座標の範囲は表3.5のとおりです．

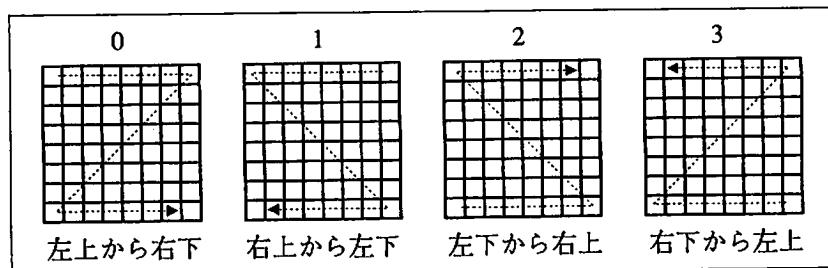



図3.1 データの転送方向

| 画面モード | X座標 | Y座標 | ページ番号 |
|---|---|---|---|
| SCREEN 5 | 0～255 | 0～191 | 0～3 |
| SCREEN 6 | 0～511 | 0～191 | 0～3 |
| SCREEN 7 | 0～511 | 0～191 | 0～1 |
| SCREEN 8 | 0～255 | 0～191 | 0～1 |
| SCREEN 10 | 0～255 | 0～191 | 0～1 |
| SCREEN 11 | 0～255 | 0～191 | 0～1 |
| SCREEN 12 | 0～255 | 0～191 | 0～1 |

表3.5　ページ番号と座標の範囲

転送する画像データは，図3.2の形式をとります．データの先頭には画像のX方向およびY方向の転送ドット数をそれぞれ2バイトデータで置いておかなければなりません．

第6パラメータはロジカルオペレーションで，表示するパターンと画面上のパターンとの間で，それぞれ図3.3に示すような演算をおこないます．

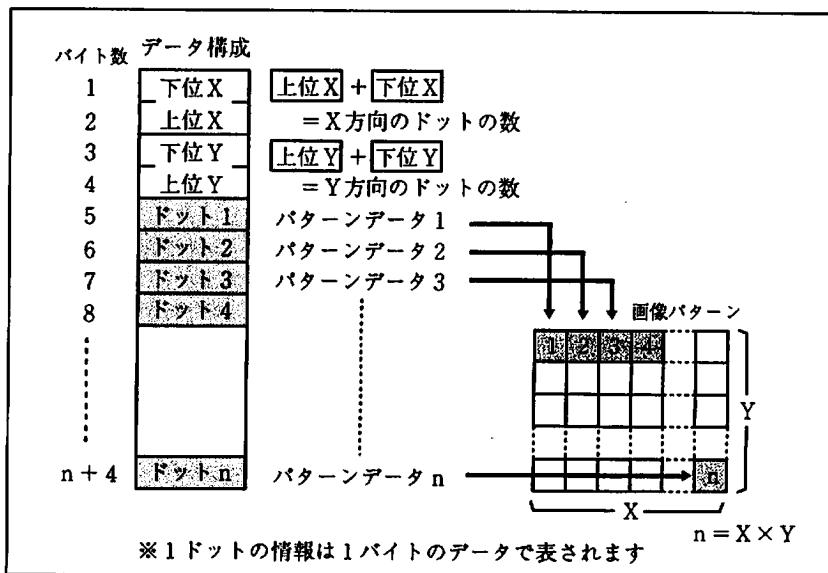



図3.2　cpym2vで表示するデータの形式

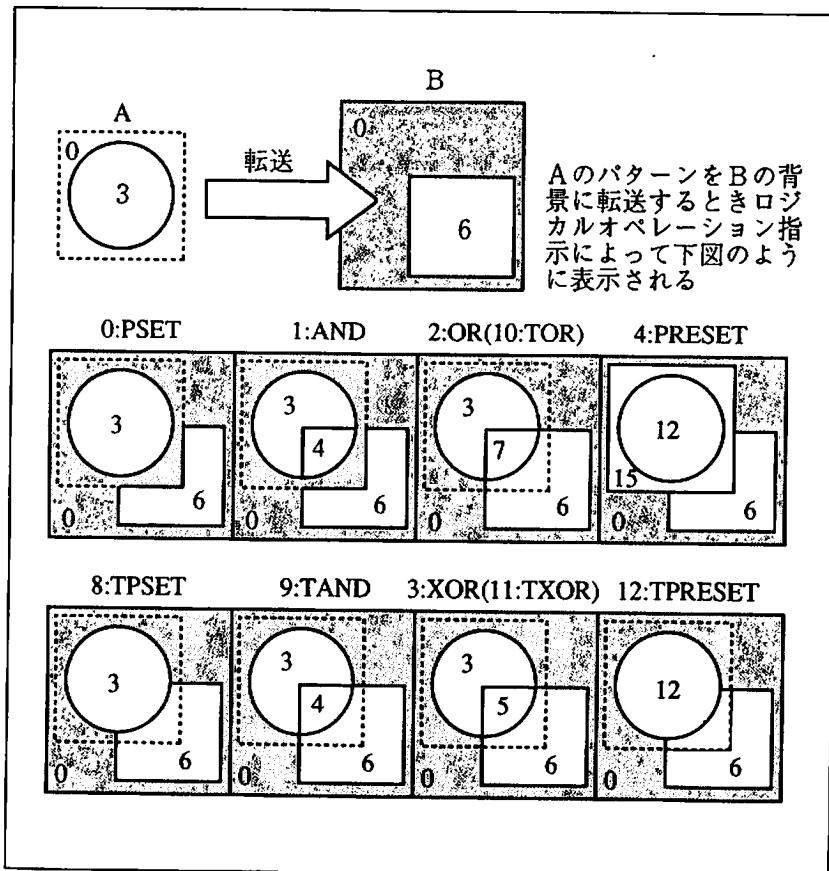



図3.3 ロジカルオペレーションの機能

【用例】

グラフィックパターンを画面中いたる所に表示します.

配列変数Ptn [] には表示するパターンが格納されています. 配列の最初の2バイトはパターンの横方向のドット数, 次の2バイトが縦方向のドット数, 残りが13×19ドットのパターンデータ本体です.

このパターンを, ロジカルオペレーション8, つまり重ね描きで, SCREEN 5の画面上に表示します. この動作は何かキーを押すまで続けられます.

```
 0: #include <stdio.h>
 1: #include <msxalib.h>
 2: #include <msxclib.h>
 3:
 4: char Ptn[4+13*19] = { 13, 0, 19, 0,     ……… パターン
 5:      0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0,  のデータ
 6:      0, 0, 4, 4, 4, 4, 4, 4, 4,15, 4, 0, 0,
 7:      0, 0, 4, 4, 4, 4, 4, 4, 4,15, 4, 0, 0,
 8:      0, 0, 4, 4, 4, 4, 4, 4, 4,15, 4, 0, 0,
 9:      0, 0, 4, 4, 4, 4, 4, 4, 4,15, 4, 0, 0,
10:      0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0,
11:      0, 0, 4, 4, 4, 4, 4, 4, 4,15, 4, 0, 0,
12:      4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
13:      0, 0, 0,11,11, 8, 8, 8,11,11, 0, 0, 0,
14:      0, 4, 4, 4,11,11,11,11,11, 4, 4, 4, 0,
15:      4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
16:      4, 4, 4, 4, 4, 4,11, 4, 4, 4, 4, 4, 4,
17:     11,11, 4, 4, 4, 4, 4, 4, 4, 4, 4,11,11,
18:     11,11, 4,11, 4,11,11,11, 4,11, 4,11,11,
19:      0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0,
20:      0, 0, 4, 4, 4, 4, 4, 4, 4, 4, 4, 0, 0,
21:      0, 8, 8, 0,11,11, 0,11,11, 0, 8, 8, 0,
22:      8, 8, 8, 8, 8, 8, 0, 8, 8, 8, 8, 8, 8,
23:      8, 8, 8, 8, 8, 8, 0, 8, 8, 8, 8, 8, 8 };
24:
25: VOID main()
26: {
27:     int i;
28:
29:     ginit();    ………………… cpym2v 実行時には ginit が必要
30:     srnd();
31:     color(15, 1, 1);
32:     screen(5);
33:
34:     while (!kbhit()) {
35:         cpym2v(Ptn, 0,
36:              rnd(240), rnd(195), 0, 8);  ….. 8:TPSET
37:         for (i = 0; i < 4000; ++i);
38:     }
39:     screen(0);
40: }
```

リスト 3.9 CPYM2V.C

# cpyv2v関数

【機能】　グラフィックパターンをVRAM上で移動します

【仕様】　#include　<msxalib.h>

```
VOID cpyv 2 v (x 1, y 1, x 2, y 2, sp, dx, dy, dp, lg)
int x 1; …… 転送元のx座標の始点
int y 1; …… 転送元のy座標の始点
int x 2; …… 転送元のx座標の終点
int y 2; …… 転送元のy座標の終点
int sp;  …… 転送元のページ番号
int dx;  …… 転送先のx座標の始点
int dy;  …… 転送先のy座標の始点
int dp;  …… 転送先のページ番号
int lg;  …… ロジカルオペレーション
```

【説明】

cpyv2v関数はSCREEN 5～12 専用（本関数はVDPを直接アクセスすることにより実現しています．したがってSCREEN 2～4 では使用できません）で，第5パラメータspが示すVRAMのページ上の座標（x 1, y 1）から（x 2, y 2）の範囲の画像を，第8パラメータdpが示すページ上の座標（dx, dy）へ転送します．なお，第9パラメータlgはロジカルオペレーションで，cpym2v関数のものと同じです．

【用例】

キャラクタが帽子の中に隠れていきます．

まずパターンを裏ページ（ページ 1）に転送してから，このパターンを表ページ（ページ 0）にも転送します．そして，ページ 1 から帽子の部分だけをペ

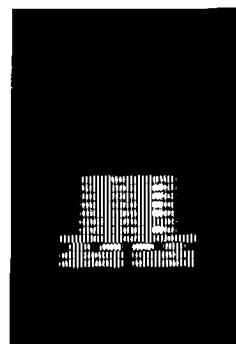
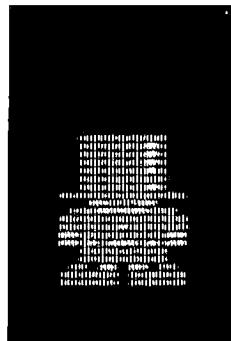
ージ0に表示されているキャラクタの上に，従来の位置よりも1ドット下げて転送します．これを2ドット下げ，3ドット下げ，最終的には8ドットまで下げ，帽子と靴だけの姿になってしまうのです．

お気付きになりましたでしょうか? ここでのポイントは転送する帽子のパターンの最上ラインにあります．実は，このラインはすべて0で構成されています．このラインは，転送されるごとに生じる画像ズレによる余分な部分を消去しているのです．言い換えれば，このラインは画面上のゴミを拭き取る作用をしているわけです．些細なことでありますが，高速処理が求められる場合には有効な一手段であり，覚えておいて損はないと思います．

```
 Ø: #include <stdio.h>
 1: #include <msxalib.h>
 2: #include <msxclib.h>
 3:
 4: char Ptn[4+13*2Ø] = { 13, Ø, 2Ø, Ø,
 5:         Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø, Ø,
 6:         Ø, Ø, 4, 4, 4, 4, 4, 4, 4, 4, 4, Ø, Ø,
 7:         Ø, Ø, 4, 4, 4, 4, 4, 4, 4,15, 4, Ø, Ø,
 8:         Ø, Ø, 4, 4, 4, 4, 4, 4, 4,15, 4, Ø, Ø,
 9:         Ø, Ø, 4, 4, 4, 4, 4, 4, 4,15, 4, Ø, Ø,
1Ø:         Ø, Ø, 4, 4, 4, 4, 4, 4, 4,15, 4, Ø, Ø,
11:         Ø, Ø, 4, 4, 4, 4, 4, 4, 4, 4, 4, Ø, Ø,
12:         Ø, Ø, 4, 4, 4, 4, 4, 4, 4,15, 4, Ø, Ø,
13:         4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
14:         Ø, Ø, Ø,11,11, 8, 8, 8,11,11, Ø, Ø, Ø,
15:         Ø, 4, 4, 4,11,11,11,11,11, 4, 4, 4, Ø,
16:         4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4, 4,
17:         4, 4, 4, 4, 4, 4,11, 4, 4, 4, 4, 4, 4,
18:        11,11, 4, 4, 4, 4, 4, 4, 4, 4, 4,11,11,
19:        11,11, 4,11, 4,11,11,11, 4,11, 4,11,11,
2Ø:         Ø, Ø, 4, 4, 4, 4, 4, 4, 4, 4, 4, Ø, Ø,
21:         Ø, Ø, 4, 4, 4, 4, 4, 4, 4, 4, 4, Ø, Ø,
22:         Ø, 8, 8, Ø,11,11, Ø,11,11, Ø, 8, 8, Ø,
23:         8, 8, 8, 8, 8, 8, Ø, 8, 8, 8, 8, 8, 8,
24:         8, 8, 8, 8, 8, 8, Ø, 8, 8, 8, 8, 8, 8 };
25:
26: VOID main()
27: {
```

```
28:     int i, j;
29:
3ø:     ginit();   ................. cpyv2v 実行時には ginit が必要
31:     color(15, 1, 1);
32:     screen(5);
33:     cpym2v(Ptn, ø, 1ø, 1ø, 1, ø);   ....... パターンを
34:                                              裏ページに転送
35:     while (!kbhit()) {
36:         for (j = ø; j < 3øøøø; ++j);
37:         cpyv2v(1ø, 1ø, 22, 29, 1, 1øø, 1øø, ø, ø);
38:         for (j = ø; j < 2øøøø; ++j);
39:         for (i = 1ø1; i <= 1ø8; ++i) {
4ø:             cpyv2v(1ø, 1ø, 22, 18, 1, 1øø, i, ø, ø);
41:             for (j = ø; j < 2øøø; ++j);
42:         }
43:     }
44:
45:     screen(ø);
46: }
```

リスト3.10　CPYV2V.C

帽子がするする降りてゆく

CPYV2V.C の実行画面

# inispr関数

【機能】 スプライトを初期化しサイズを決定します

【仕様】 #include 〈msxclib. h〉

```
VOID inispr (sz)
int sz; …… スプライトサイズ番号
```

【説明】

inispr () 関数は，スプライト表示を初期化するとともに，スプライトのサイズを定義します．スプライトサイズは図3.4のとおり4種類用意されています．

【用例】

putspr関数の項を参照してください．

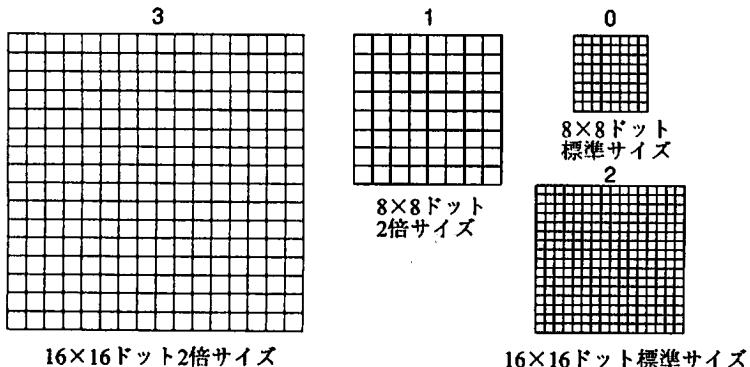



図3.4　スプライトのサイズの設定

## sprite関数

【機能】 スプライトパターンを定義します

【仕様】 #include 〈msxclib. h〉

```
VOID sprite (no, dt)
int no;      …… スプライトパターン番号
char *dt;…… スプライトパターンデータへのポインタ
```

【説明】

sprite関数は，第1パラメータnoで指定される番号のスプライトに，第2パラメータdtで示されるスプライトパターンデータを定義します．スプライトパターンは，8×8サイズのときは最大256個まで，16×16サイズのときは最大64個まで登録することができます．

まず，8×8サイズのスプライトパターンの構造について，簡単に述べていきます．8×8サイズのスプライトでは，図3.5のように，画面に表示するドットを1，透明のままにしておくドットを0と考えると，横8ドットは8桁の

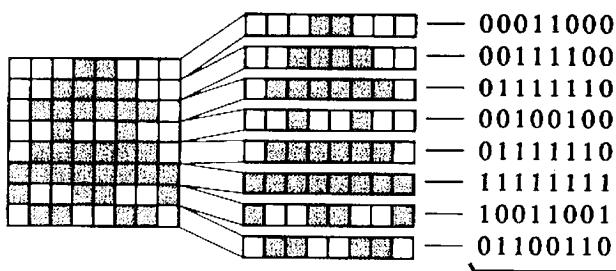



図3.5　8×8ドットサイズのスプライトパターンデータ

2進数，つまりchar型のデータとして表記することができます．これを縦8ドットぶん繰り返すと，char型のデータ8個で8×8ドットのスプライトパターンデータが表されます．この8×8サイズのスプライトパターンは最大256個まで登録することができ，それぞれ0～255のスプライトパターン番号を持っています．

16×16ドットサイズのスプライトパターンは，8×8ドットのスプライ4枚を図3.6のように並べたものです．したがって，8×8ドットのスプライトパターンデータ8個を1,2,3,4と順々に並べた32個のchar型データで，16×16サイズのスプライトパターンが定義されます．16×16サイズのスプライトパターンは最大64個まで登録することができ，それぞれ0～255の下位2ビットを無視した，つまり0,4,8,……のスプライトパターン番号で指定されます．

【用例】

putspr関数の項を参照してください．

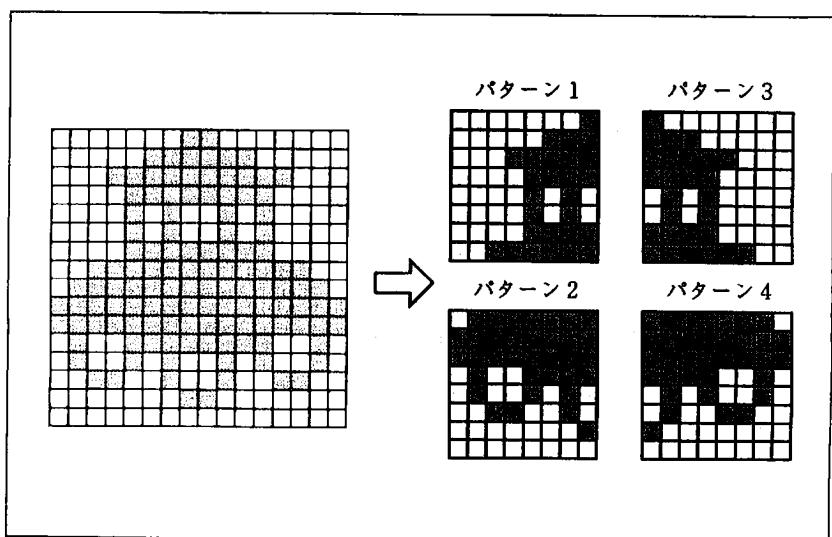



図3.6　16×16ドットサイズのスプライトパターン構成

# putspr関数

【機能】 スプライトを画面に表示します

【仕様】 #include 〈msxclib. h〉

```
VOID putspr (no, x, y, cl, pt)
int no; …… スプライト面番号 (0～31)
int x;  …… x座標 (0～255)
int y;  …… y座標 (-16～190 または-16～210)
int cl; …… カラー番号あるいはパレット番号
int pt; …… スプライトパターン番号 (0～255)
```

【説明】

putspr関数は，第1パラメータnoで指定されるスプライト面の座標 (x, y) に，第4パラメータclで指定される色で，第5パラメータptで指定される番号のスプライトパターンを表示します．ただし，モード2のスプライト (SCREEN 4～8) では，clで指定した色は無視されます．モード2のスプライトは，次に紹介するcolspr関数を使って色を設定してください．

MSXのスプライト表示におけるy座標 (縦座標) は-1から始まる系列を持っています．つまり，驚くなかれ通常y座標0である画面最上段は，なんとスプライトにおいてはy座標-1という世にも奇怪な現象となっているのです．

スプライトモード1の場合はy座標を208，スプライトモード2の場合はy座標を216に指定することで，そのスプライト面以降のスプライトはまったく表示されなくなります．

なお，このputspr関数では，x座標にマイナスの値を与えてスプライトを画面の左端へ消し去ることはできません．それを実現するには，かなり面倒な処理が必要で，スプライト表示の速度にも影響が出てくるため，機能を割愛させてもらいました．

【用例】

ボールが放物線を描きながら画面上ではずみます.

まずinispr関数を実行して，スプライトを16×16サイズに設定します．次にsprite関数で，配列Sprpt［］に用意したデータを0番のスプライトパターンとして与えます.

putspr（0,x,y,7,0）は，スプライト面0の座標（x,y）に，カラー7で，0番のスプライトを表示します．putspr（1,240-x,y,8,0）は，スプライト面1の座標（240-x,y）に，カラー8で，0番のスプライトを表示します.

ここではSCREEN 2を使っているため，スプライトのモードは1となり，putspr関数の第4パラメータでスプライトの色が指定できます．モード2のスプライトを利用する場合の色の指定方法は，colspr関数の項を参照してください.

```
 0: #include <stdio.h>
 1: #include <msxalib.h>
 2: #include <msxclib.h>
 3:
 4: char Ptspr[32] = {   ................ ボールのスプライトパターン
 5:      0x07, 0x1f, 0x37, 0x7f, 0x6f, 0xdf, 0xdf, 0xdf,
 6:      0xff, 0xff, 0xff, 0x7f, 0x7f, 0x3f, 0x1f, 0x07,
 7:      0xe0, 0xf8, 0xfc, 0xfe, 0xfe, 0xff, 0xff, 0xff,
 8:      0xff, 0xff, 0xff, 0xfe, 0xfe, 0xfc, 0xf8, 0xe0 };
 9:
10: VOID main()
11: {
12:      int x, y, dx, dy, i;
13:
14:      color(15, 1, 1);
15:      screen(2);
16:      inispr(2);              ........ 16×16 ドット通常サイズを指定
17:      sprite(0, Ptspr);       ........ スプライトパターンの設定
18:
19:      for (x=0, y=4, dx=3, dy=0; y<175 || dy; ++dy) {
20:          putspr(0, x, y, 7, 0);          ........ ボールを第0面に表示
21:          putspr(1, 240-x, y, 8, 0);      ........ ボールを第1面に表示
22:          if ((x += dx) == 240 || x == 0) dx = -dx;
23:          if ((y += dy) == 175) dy = -dy;
24:          for (i = 0; i < 1000; ++i);
25:      }
26:
27:      screen(0);
28: }
```

リスト3.11　PUTSPR.C

# colspr関数

【機能】 スプライトの1ラインごとの色を指定します

【仕様】 #include 〈msxclib. h〉

VOID colspr (no, dt)
int no; ……スプライト面番号
char ＊dt; ……スプライトカラーデータへのポインタ

【説明】

この関数は，モード2 (SCREEN 4~8) のスプライトを使用する場合のみ意味を持ちます．モード2のスプライトは，1ラインごとに異なる色を指定できます．これでモード1のスプライトの欠点の1つである乏しい色彩表現が多少なりとも克服されるわけです．

色の指定には8×8サイズの場合は8個のカラーデータ，16×16サイズの場合は16個のカラーデータが必要となります．カラーデータの構造は図3.7

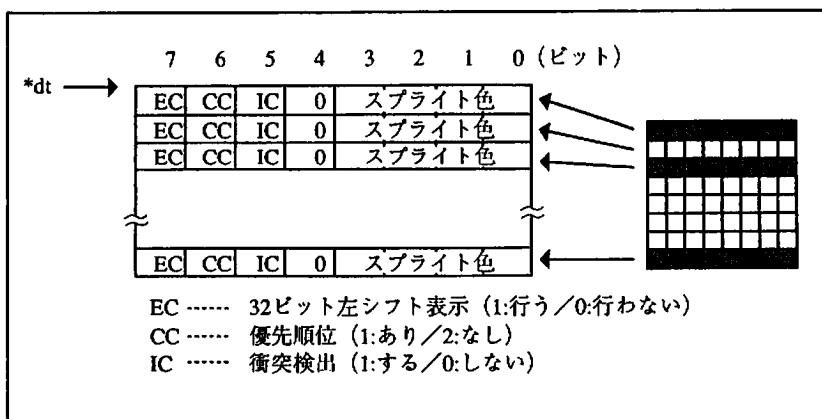



図3.7 スプライトカラーテーブルの構造

| 0 | 黒 | 4 | 暗い緑 | 8 | 肌色 | 12 | 緑 |
|---|---|---|---|---|---|---|---|
| 1 | 暗い青 | 5 | 暗い水色 | 9 | 青 | 13 | 水色 |
| 2 | 暗い赤 | 6 | 暗い黄色 | 10 | 赤 | 14 | 黄色 |
| 3 | 暗い紫 | 7 | 灰色 | 11 | 紫 | 15 | 白 |

表3.6　SCREEN 8のスプライトの色

に示すとおりです．スプライトの色は基本的にはパレット番号で指定します．ただしSCREEN 8では表3.6の16色に固定されています．

ここで奥義「スプライト2枚重ね」を披露しましょう．モード2のスプライトでは，2枚のスプライトパターンを使うことで，横1ラインにつき3色まで出すことができるのです．

どうしてスプライト2枚で3つの色が出せるのでしょうか．通常はスプライトが重なると，手前にあるスプライトが後ろのスプライトを隠してしまいます．ところが，後ろのスプライトのカラーテーブルのCCビットを"1"にしておくと，そのラインの重なった部分は，両者のカラーコードのOR(論理和)をとったカラーコードで表示されるのです．ですから，スプライトを2枚使うだけで図3.8のように，(1) 手前のスプライトの色，(2) 後ろのスプライトの色，(3) 重なった部分の色，という3色が出せるわけです．

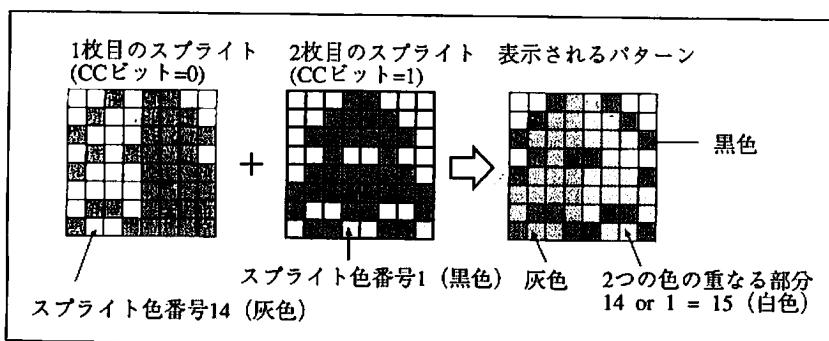



図3.8　スプライト2枚重ね

実際には重ねるスプライトは何枚でもかまいません．その場合一番手前のスプライトだけCCビットを“0”にしておいて，残りのスプライトはCCビットをすべて“1”に設定します．ただし，あまりたくさん使うと今度はスプライトの表示に手間がかかりますし，「1ライン上に8枚まで」というスプライトの表示枚数制限にもひっかかりやすくなりますから，やはり実用的には2枚重ねまでが限度でしょう．

**【用例】**

画面上を蝶が舞います．

このプログラムではスプライト2枚重ねを使っています．スプライトパターンは，配列Ptspr 0 [] に0番のスプライト，配列Ptspr 1 [] に1番のスプライトのデータを入れています．スプライトカラーは，配列Clspr 0[]が0番のスプライト，配列Clspr 1 [] が1番のスプライト用です．Clspr 1 [] のスプライトカラーデータはすべて，CCビットを“1”にした場合の加算値64を本来のパレット番号に加えた値になっています．これが，スプライト2枚重ねの秘訣であります．

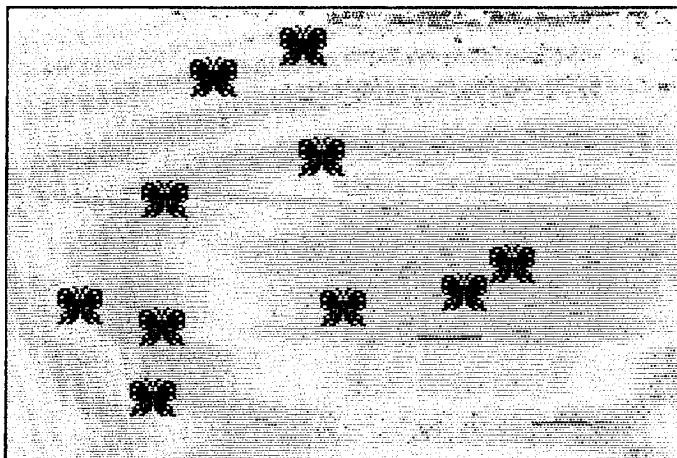
move関数はスプライトを移動させます．蝶が宙を舞うがごとくみせるために，x座標そしてy座標それぞれを-1～+1のランダムなごくわずかな移動の連続で表しています．そして，x座標は0～240の範囲で，y座標は-1～195の範囲で移動させています．

```
 0: #include <stdio.h>
 1: #include <msxalib.h>
 2: #include <msxclib.h>
 3:
 4: char Ptspr0[32] = {    ………… 1枚目のスプライトパターン
 5:      0x04, 0x72, 0xfa, 0xfc, 0xfe, 0xfe, 0xfe, 0xfe,
 6:      0x72, 0x1a, 0x36, 0x76, 0x7e, 0x7c, 0xd8, 0x80,
 7:      0x40, 0x9c, 0xbe, 0x7e, 0xfe, 0xfe, 0xfe, 0xfe,
 8:      0x9c, 0xb0, 0xd8, 0xdc, 0xfc, 0x7c, 0x36, 0x02 };
 9:
10: char Ptspr1[32] = {    ………… 2枚目のスプライトパターン
11:      0x00, 0x00, 0x20, 0x01, 0x43, 0x01, 0x01, 0x41,
```

```
12:      øxød, øxø5, øxø9, øx28, øxøø, øx1ø, øxøø, øxøø,
13:      øxøø, øxøø, øxø8, øxøø, øx84, øxøø, øxøø, øxø4,
14:      øx6ø, øx4ø, øx2ø, øx28, øxøø, øx1ø, øxøø, øxøø };
15:
16: char Clsprø[16] = {   ………… 1枚目のスプライトの色
17:      1, 1, 1, 1, 1, 1, 1, 1,
18:      1, 1, 1, 1, 1, 1, 1, 8 };
19:
2ø: char Clspr1[16] = {   ………… 2枚目の色(すべて+64)
21:      12+64,12+64,12+64,12+64,12+64, 6+64, 6+64,12+64,
22:      12+64,12+64,12+64,12+64,12+64,12+64,12+64,12+64};
23:
24: int Xpos[12], Ypos[12];
25:
26: VOID move()
27: {
28:     int i, x, y;
29:
3ø:     for (i = ø; i < 12; ++i) {
31:         x = Xpos[i] + rnd(3) - 1;
32:         y = Ypos[i] + rnd(3) - 1;
33:         if (x < ø)   x = ø;
34:         if (x > 24ø) x = 24ø;
35:         if (y < -1)  y = -1;
36:         if (y > 195) y = 195;
37:         putspr(i * 2, x, y, ø, ø);
38:         putspr(i * 2 + 1, x, y, ø, 4);
39:         Xpos[i] = x;
4ø:         Ypos[i] = y;
41:     }
42: }
43:
44: VOID main()
45: {
46:     int i;
47:
48:     srnd();
49:     ginit();
5ø:     color(15, 1ø, 1ø);
51:     palett(13, 4, 7, 5);
52:     screen(5);   ………… SCREEN 5ではスプライトのモードは2
53:     inispr(2);
```

```
54:     sprite(0, Ptspr0);
55:     sprite(4, Ptspr1);
56:
57:     for (i = 0; i < 12; ++i) {
58:         Xpos[i] = rnd(120) + 60;
59:         Ypos[i] = rnd(120) + 38;
60:         colspr(i * 2, Clspr0);         ····· 通常表示
61:         colspr(i * 2 + 1, Clspr1);     ····· Clspr1 のデータに
62:     }                                        64 が足されている
63:                                              ので、重ね表示
64:     while (!kbhit()) {
65:         move();
66:         for (i = 0; i < 400; ++i);
67:     }
68:
69:     color(15, 1, 1);
70:     palett(13, 6, 2, 5);
71:     screen(0);
72: }
```

リスト 3.12　COLSPR.C

スプライト 2 枚重ねの極彩色蝶

COLSPR.C の実行画面

# vpoke関数

【機能】　VRAMへ１バイトのデータを書き込みます（ginit() が必要）

【仕様】　#include　<msxalib.h>

```
VOID vpoke (ad, dt)
unsigned ad;   …… VRAMアドレス
char dt;       …… 書き込むデータ
```

【説明】

vpoke関数は，第１パラメータadの指し示すVRAMアドレスに，第２パラメータの指し示す１バイト（char型）のデータdtを書き込みます．

画面にデータを表示する場合に使えるのはもちろんですが，MSXでは実際に画面に表示されるデータ以外にも，フォントパターンやカラーテーブルがVRAM上に存在しています．このvpoke関数をはじめ，これから紹介するvpeek関数，ldirmv関数，ldirvm関数は，こういったVRAM上のデータを操作する場合にも利用できます．

【用例】

渦を巻くように画面の外から内に文字を表示し消去します．

プログラムの最初に定義しているfill関数は，SCREEN 1の画面上にパラメータchで示されるキャラクタコードを次々と書き込んでいきます．なお，SCREEN 1の画面のVRAMアドレスは，メインメモリの0xf3bd番地に記録されています．

最初のfor文の中に４組のfor文が存在しますが，それぞれ左から右へ，上から下へ，右から左へ，下から上へと文字を書き込みます．

```
Ø: #include <stdio.h>
1: #include <msxalib.h>
2: #include <msxclib.h>
3:
4: #define T32NAM *((int *)Øxf3bd)    ………SCREEN 1の先頭アドレス
5:
6: VOID fill(ch)
7: char ch;
8: {
9:     int i, j, k;
1Ø:     int ad = T32NAM - 1;
11:     int x = 32;
12:     int y = 23;
13:
14:     for (i = Ø; i <= 11; ++i) {
15:         for (j = Ø; j < x; ++j) {    ………左から右に向かって表示
16:             vpoke(++ad, ch);
17:             for (k = Ø; k < 1ØØØ; ++k);
18:         }
19:         --x;
2Ø:         for (j = Ø; j < y; ++j) {    ………上から下に向かって表示
21:             vpoke(ad += 32, ch);
22:             for (k = Ø; k < 1ØØØ; ++k);
23:         }
24:         --y;
25:         for (j = Ø; j < x; ++j) {    ………右から左に向かって表示
26:             vpoke(--ad, ch);
27:             for (k = Ø; k < 1ØØØ; ++k);
28:         }
29:         --x;
3Ø:         for (j = Ø; j < y; ++j) {    ………下から上に向かって表示
31:             vpoke(ad -= 32, ch);
32:             for (k = Ø; k < 1ØØØ; ++k);
33:         }
34:         --y;
35:     }
36: }
37:
38: VOID main()
39: {
4Ø:     int i;
41:
42:     ginit();              ………vpoke実行時にはginitが必要
43:     color(15, Ø, Ø);
44:     screen(1);
45:     fill('?');            ………画面を'?'で埋める
46:     fill(' ');            ………画面を' 'で埋める(つまり文字を消す)
47:     screen(Ø);
48: }
```

リスト3.13 VPOKE.C

# vpeek関数

【機能】 VRAMから１バイトのデータを読み出します（ginit() が必要）

【仕様】 #include <msxalib.h>

```
char vpeek (ad)
unsigned ad;      …… VRAMアドレス
```

【説明】

vpeek () 関数は，パラメータadの指し示すVRAMアドレスの内容１バイト（char型）のデータを返します．

【用例】

テキスト画面の各ラインを左右反転させます．

まず，現在使用しているテキスト画面のVRAMアドレスと１行の長さを調べて，それぞれadtop, lenという変数に記憶します．続いて，画面の各ラインについて文字の並び方を左右ひっくり返します．

現在の画面モードはメインメモリ上の0xfcaf番地に記録されています．現在の１行中の文字数は同じく 0xf3b0 番地に記録されています．SCREEN 0 の画面先頭アドレスは 0xf3b3 番地，SCREEN 1 の画面先頭アドレスは 0xf3bd番地にあります．

```
0: #include <stdio.h>
1: #include <msxalib.h>
2:
3: #define SCRMOD *((char *)0xfcaf)       ..... スクリーンモード
4: #define LINLEN *((char *)0xf3b0)       ..... 1行の長さ
5: #define TXTNAM *((unsigned *)0xf3b3) ..... SCREEN0 の先頭
6: #define T32NAM *((unsigned *)0xf3bd) ..... SCREEN1 の先頭
```

```
 7:
 8: VOID main()
 9: {
10:     unsigned  i, a1, a2, adtop, len;
11:     char ch1, ch2;
12:
13:     ginit();                  ........ vpeek実行時にはginitが必要
14:     if (SCRMOD == 0) {        ........ SCREEN 0の場合
15:         adtop = TXTNAM;
16:         len = (LINLEN > 40)? 80: 40;   .... LINLEN>40なら
17:     }                                       80字モード
18:     else {                    ........ SCREEN 1の場合
19:         adtop = T32NAM;
20:         len = 32;
21:     }
22:     for (i = 0; i < 24; ++i) {
23:         a1 = adtop + len * i;     ........ 行の左端のアドレス
24:         a2 = a1 + len - 1;        ........ 行の右端のアドレス
25:         do {
26:             ch1 = vpeek(a1);
27:             ch2 = vpeek(a2);
28:             vpoke(a1, ch2);
29:             vpoke(a2, ch1);
30:         } while (++a1 < --a2);
31:     }
32: }
```

リスト3.14 VPEEK.C

《MSX-C豆知識》 #defineマクロについて

「#define」は文字列の置き換えをおこなう置換マクロです．たとえば，「0xf3bd番地に記録されているunsigned型のデータ」は，キャスト演算子とポインタ演算子を使って，「* ((unsigned *) 0xf3bd)」と表しますが，これをそのままプログラムの中に書くとゴチャゴチャしてわかりにくくなってしまいます．そこで#defineマクロを使って，これを「T 32 NAM」と定義しておくと，以後ソースプログラムに現れる「T 32 NAM」は，すべてその文字列に置き換えられることになり，プログラムが見やすくなります．

# ldirvm関数

【機能】 VRAMへ一度に複数のデータを書き込みます（ginit() が必要）

【仕様】 #include　<msxalib.h>

```
VOID ldirvm (ds, sr, ln)
unsigned ds;   …… 転送先のVRAMアドレス
char *sr;      …… 転送元データへのポインタ
unsigned ln;   …… 転送データ長
```

【説明】

第1パラメータdsが示すVRAMアドレスへ，第2パラメータsrが示すアドレスのメインメモリから，第3パラメータlnが示すバイト数だけデータを転送します．

【用例】

フォントパターンを書き換えます．まず画面に〝0″から〝9″までの数字が表示されます．ここで何かキーを押すと，そのフォントが変更されます．もう一度何かキーを押すとプログラムは終了します．

SCREEN 1のフォントパターンは，0xf3c1番地に記録されているVRAMアドレスから，1文字につき8バイトを使って格納されています．

2次元配列data [10] [8] は，10個ぶんのフォントデータの定義です．このデータを数字〝0″のフォントパターンのアドレスに転送することで，〝0″から〝9″の10個の数字のフォントパターンを書き換えることができます．

なお，screen関数を実行すると，フォントパターンは元に戻ります．

```
 0: #include <stdio.h>
 1: #include <msxalib.h>
 2: #include <msxclib.h>
 3:
 4: #define T32CGP (*(unsigned *)0xf3c1)  …… フォント
 5:                                          のアドレス
 6: char data[10][8] = {
 7:     { 0x00,0x00,0x00,0xf0,0x90,0x90,0xf0,0x00 },
 8:     { 0x60,0x20,0x20,0x20,0x20,0x20,0x20,0x00 },
 9:     { 0xf8,0x08,0x08,0xf8,0x80,0x80,0xf8,0x00 },
10:     { 0xf8,0x88,0x08,0xf8,0x08,0x88,0xf8,0x00 },
11:     { 0x90,0x90,0x90,0x90,0xf8,0x10,0x10,0x00 },
12:     { 0xf0,0x80,0x80,0xf8,0x08,0x08,0xf8,0x00 },
13:     { 0x80,0x80,0x80,0xf8,0x88,0x88,0xf8,0x00 },
14:     { 0xf8,0x88,0x10,0x10,0x20,0x20,0x20,0x00 },
15:     { 0xf8,0x88,0x88,0xf8,0x88,0x88,0xf8,0x00 },
16:     { 0xf8,0x88,0x88,0xf8,0x08,0x08,0x08,0x00 } };
17:
18: VOID main()
19: {
20:     int i;
21:
22:     ginit();    ………………… ldirvm 実行時には ginit が必要
23:     screen(1);
24:     puts("¥f0123456789¥n");
25:     getch();
26:     ldirvm(T32CGP + '0' * 8, data, 80);  ….. '0' から
27:     getch();                                10 文字の
28:     screen(0);                              フォントを
29: }                                           変更
```

リスト 3.15　LDIRVM.C

---

《MSX-C豆知識》 フォントパターンのアドレス

フォントパターンが存在するVRAM上のアドレスは，SCREEN 0 と SCREEN 1 で，それぞれ次のワークエリアに記録されています．

SCREEN 0 : 0xf3b7

SCREEN 1 : 0xf3c1

# ldirmv関数

【機能】 VRAMから一度に複数のデータを読み出します (ginit() が必要)

【仕様】 #include <msxalib.h>

```
VOID ldirmv (ds, sr, ln)
char *ds;       …… 転送先データ領域へのポインタ
unsigned sr;    …… 転送元のVRAMアドレス
unsigned ln;    …… 転送データ長
```

【説明】

第1パラメータsrが示すメインメモリへ，第2パラメータsrが示すVRAMアドレスから，第3パラメータlnが示すバイト数だけデータを転送します．

【用例】

フォントパターン "A" を拡大表示します．

まずフォントパターン "A" をldirmv関数によって読み取ります．読み込んだフォントパターンはchar型のデータ1つが横8ドットのパターン構成となっています．

そして，ビットシフト演算子を用いながら，bit 7が "0" のときはスペース，"1" の時は "A" を画面上に描いていきます．これを縦8ラインぶん繰り返すことで，画面上にフォントパターン "A" を拡大して表示します．

```
Ø: #include <stdio.h>
 1: #include <msxalib.h>
 2: #include <msxclib.h>
 3:
 4: #define T32CGP *((int *)Øxf3c1)
 5:
 6: VOID main()
 7: {
 8:     char data[8];
 9:     int i, j;
1Ø:
11:     ginit();    ………………… ldirmv 実行時には ginit が必要
12:     screen(1);
13:     ldirmv(data, T32CGP + 'A' * 8, 8);  ….. 'A' フォント
14:                                          を読み込む
15:     for (i = Ø; i < 8; ++i) {
16:         for (j = Ø; j < 8; ++j) {
17:             if ((data[i] & Øx8Ø) == Ø)  ….. 左端ビット
18:                 putchar(' ');                は 0 か？
19:             else
2Ø:                 putchar('A');
21:             data[i] <<= 1;   ….. data[i] を左にシフト
22:         }
23:         putchar('¥n');
24:     }
25:
26:     getch();
27:     screen(Ø);
28: }
```

リスト 3.16 LDIRMV.C

# filvrm関数

【機能】 VRAM上の連続範囲を一定のデータで埋めます（ginit() が必要）

【仕様】

```
#include <msxalib.h>
VOID filvrm (ad, ln, dt)
unsigned ad;  ...... VRAMアドレス
unsigned ln;  ...... 書き込む長さ
char dt;      ...... 書き込むデータ
```

【説明】

第1パラメータadが示すVRAMアドレスから，第2パラメータlnが示す長さだけ，第3パラメータdtが示すデータでVRAMを埋めます．

【用例】

SCREEN 5で特殊効果的な画面消去をおこないます．

プログラムの最初に定義しているerase関数は，SCREEN 5の画面上で，8行おきの0, 8, 16, ・・・, 208の各ラインに相当するVRAMにデータ0を書き込みます．しばらく待機してから，その1つ下の系列1, 9, 17, ・・・, 209の各ラインを消去していきます．これを8回繰り返すことで，画面はすべてクリアされます．

```
Ø: #include <stdio.h>
 1: #include <msxalib.h>
 2: #include <msxclib.h>
 3:
 4: VOID erase()
 5: {
 6:     int i, j;
 7:
 8:     for (i = Ø; i <= 7; ++i) {
 9:         for (j = i; j <= 211; j += 8)
1Ø:             filvrm(j * 128, 128, (char)Ø);  ····· グラフィック
11:         for (j = Ø; j < 7ØØØ; ++j);           画面は必ず VRAM
12:     }                                         0 番地から始まる
13: }                                             ため J*128 には
14:                                               何も加えなくて
15: VOID main()                                   よい
16: {
17:     int i;
18:
19:     ginit();    ·················· filvrm 実行時には ginit が必要
2Ø:     srnd();
21:     color(15, 1, 1);
22:     screen(5);
23:     for (i = Ø; i <= 4Ø; ++i)
24:       boxfil(rnd(256),rnd(212),rnd(256),rnd(212),rnd(15),Ø);
25:     for (i = Ø; i < 3ØØØØ; ++i);
26:     erase();
27:     for (i = Ø; i < 3ØØØØ; ++i);
28:     screen(Ø);
29: }
```

リスト 3.17　FILVRM.C

# snsmat関数

【機能】　押されたキーを即座に読み取ります

【仕様】　#include　<msxclib.h>

```
TINY snsmat (mt)
int mt; …… 読み出すキーマトリクスの行
```

【説明】

MSX-C標準ライブラリには，getch関数，getche関数，gets関数の3つの文字入力用関数が用意されています．これらの関数は非常に有効な関数ですが，タイマー割り込み禁止中は使用できませんし，キーボードバッファに保存されるためにタイムラグがある（つまり以前押したキーでまだ読み取っていないものを現在のキー入力として読み込んでしまう）などという欠点を持っています．

そこで登場するのがこのsnsmat関数です．この関数は，MSX BIOSの0141hをコールすることで実現しています．

MSXはキー入力の有無を確認するために図3.9のようなキーマトリクスを用います．読み出すキーマトリクスの行をパラメータとしてこの関数に引き渡すと，その行の現時点の状態がTINY型のデータとして戻ってきます．押されているキーに対応したビットは "0"，押されていないキーに対応したビットは "1" になります．

AND演算子「&」，OR演算子「｜」，ビットシフト演算子「>>」などを利用すると，今現在どのキーが押されているかの判定ができます．複数キーが同時に押されていることも判定可能です．

【用例】

移動するスプライトが，STOPキーを押すと停止し，もう一度STOPキーを

| 行 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 (ビット) |
|---|---|---|---|---|---|---|---|---|
| 0h | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1h | ; | [ | @ | ¥ | ^ | - | 9 | 8 |
| 2h | B | A | _ | / | . | , | ] | : |
| 3h | J | I | H | G | F | E | D | C |
| 4h | R | Q | P | O | N | M | L | K |
| 5h | Z | Y | X | W | V | U | T | S |
| 6h | F3 | F2 | F1 | かな | CAPS | GRAPH | CTRL | SHIFT |
| 7h | RETURN | SELECT | BS | STOP | TAB | ESC | F5 | F4 |
| 8h | → | ↓ | ↑ | ← | DEL | INS | CLS | SPACE |

図3.9 キーマトリクス

押すと再び動き出します．リターンキーを押すとプログラムは終了します．

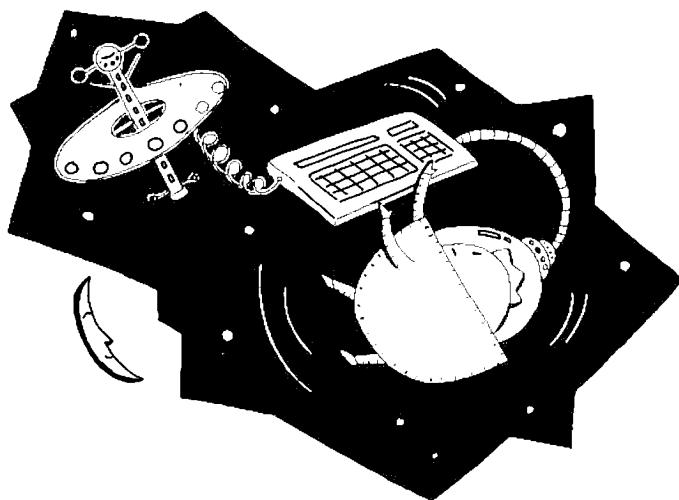
STOPキーはキーマトリクスの7行めのビット4に対応していますから，snsmat (7) と 0x10 のビットごとの論理積 (AND) をとって，それが0ならば押されていると判断できます．

プログラムの最初で定義しているstop関数は，STOPキーが押された瞬間をとらえるものです．「STOPキーが押された瞬間」というのは，「前にSTOPキーが押されていず今は押されている」状態です．stop関数では，まず現在のSTOPキーの状態を調べ，それを変数sに記録します（押されていれば1，押されていなければ0)．前回のSTOPキーの状態は変数old_sに記録してあります．そこで「s && !old_s」という条件を調べると「キーが押された瞬間」がわかるわけです．

main関数では，スプライトを左から右に動かし，STOPキーが押されるとそれを止め，もう一度STOPキーが押されるまで待ちます．プログラムを終了させるのはリターンキーですが，これはキーマトリクスの7行めのビット7に対応していますから，snsmat (7) と 0x80 のビットごとの論理積をとることで判定できます．

```
 0: #include <stdio.h>
 1: #include <msxclib.h>
 2:
 3: char Ptspr0[32] = {
 4:      0x0f,0x3f,0x63,0x41,0x41,0x41,0x63,0x7f,
 5:      0x4f,0x1e,0x01,0x07,0x00,0x00,0x00,0x00,
 6:      0xc0,0xe0,0xf0,0xf0,0xf8,0xf8,0x78,0x78,
 7:      0x78,0xf0,0xf4,0xf4,0xf6,0x6e,0x3e,0x1c };
 8:
 9: char Ptspr1[32] = {
10:      0x00,0x00,0x00,0x00,0x18,0x18,0x00,0x00,
11:      0x00,0x00,0x00,0x00,0x07,0x07,0x03,0x01,
12:      0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,
13:      0x00,0x00,0x00,0x00,0x00,0x80,0xc0,0xe0 };
14:
15: char Ptspr2[32] = {
16:      0x00,0x00,0x00,0x00,0x0f,0x3f,0x7f,0xff,
17:      0xfc,0xff,0xf7,0xfb,0xfb,0x7b,0x77,0x3f,
18:      0x00,0x00,0x00,0x00,0x80,0xe0,0xf8,0xfc,
19:      0x7c,0xbe,0xde,0xde,0xde,0x5e,0x5e,0xbc };
20:
21: TINY stop()
22: {
23:      static TINY old_s = 0;     ..... staticなので次の
24:      TINY s, r;                       呼び出しまで値が残る
25:
26:      s = ((snsmat(7) & 0x10) == 0);    ...... STOPは
27:      r = (s && !old_s);    .......... 今押されたところか
28:      old_s = s;
29:      return r;
30: }
31:
32: VOID main()
33: {
34:      char x = 0;
35:      int i;
36:
37:      color(15, 1, 1);
38:      screen(1);
39:      inispr(2);
40:      sprite(0, Ptspr0);
```

```
41:     sprite(4, Ptspr1);
42:     sprite(8, Ptspr2);
43:
44:     while ((snsmat(7) & Øx8Ø) != Ø) {  ····· RETURN?
45:         if (stop()) {
46:             putspr(Ø, x, 15Ø, 4, 8);
47:             putspr(1, x, 216, Ø, Ø);
48:             while (!stop());
49:         }
5Ø:         x -= 2;
51:         putspr(Ø, x, 15Ø, 4, Ø);
52:         putspr(1, x, 15Ø, 15, 4);
53:         for (i = Ø; i < 2ØØØ; ++i);
54:     }
55:
56:     screen(Ø);
57: }
```

リスト3.18 SNSMAT.C

# kilbuf関数

【機能】 キーバッファをクリアします.

【仕様】 #include <msxclib.h>

```
VOID kilbuf ()
```

【説明】

キーボードからの入力はタイマー割り込みで1/60秒毎にキースキャンされ，その文字コードがキーバッファに格納されていきます. kilbuf関数は，このキーバッファに格納されているデータをすべてクリアする関数です. この関数は，MSX BIOSの0156 hをコールすることで実現しています.

【用例】

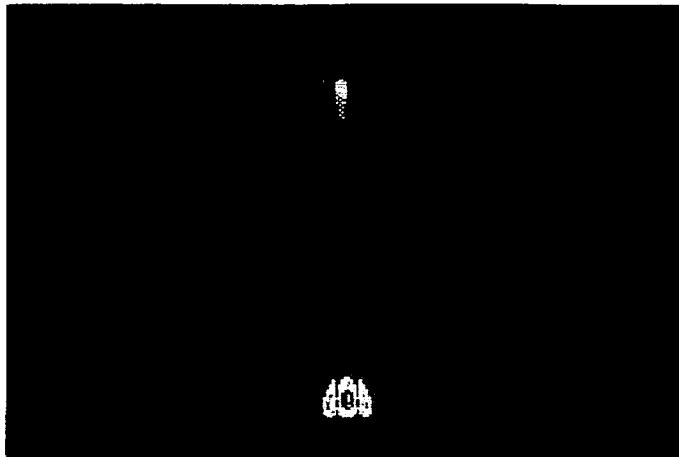
キーバッファをクリアして，キーバッファにたまった不要な文字が画面に表示されることを防ぎます.

このプログラムを実行すると，「Type some words !」というメッセージを出力します. このメッセージの出力中にいろいろキーを押し，最後にリターンキーを押してプログラムを終了させてください. そしてMSX-DOSに戻っても，画面にはなんの文字も表示されません. これがkilbuf関数の効果です.

といっても理解しづらいでしょうから，こんどは「kilbuf () ;」を削除してコンパイルし，同様に実行してみてください. すると，MSX-DOSに戻った瞬間に，それまでに入力したキーデータ群が画面にドッと表示されたと思います. このようなことを避けるために使うのがkilbuf関数です.

```
0: #include <stdio.h>
1: #include <msxclib.h>
2:
3: VOID main()
4: {
5:      puts("Type some words!¥n");
6:      puts(" and hit [RETURN] key to exit.¥n");
7:      while ((snsmat(7) & 0x80) != 0);    ····· RETURN 待ち
8:      kilbuf();   ·········· これでキーバッファがクリアされる
9: }
```

リスト3.19　KILBUF.C

トリガーボタンでミサイルを連射する

GTTRIG.C(p.83) の実行画面

# gtstck関数

【機能】 ジョイスティックの状態を読み取ります

【仕様】 #include　<msxclib.h>

```
TINY gtstck (no)
int no;          …… ジョイスティック番号
```

【説明】

ジョイスティック番号で指定した装置（ジョイスティックあるいはカーソルキー）の押されている方向を調べます．この関数は，MSX BIOSの00D5hをコールすることで実現しています．ジョイスティック番号と装置の対応は以下のとおりです．

0：カーソルキー
1：ジョイスティック1
2：ジョイスティック2

押されている方向は，0～8のTINY型の値として返されます．その戻り値と方向の対応は，図3.10に示すとおりです．

【用例】

カーソルキーまたはジョイスティックの操作に応じて，戦闘機を移動させます．ジョイスティックはポート1に接続されているものとします．リターンキーを押すとプログラムは終了します．

move関数が戦闘機の実際の移動処理となります．gtstck(0)はカーソルキーの状態を読み取り，gtstck (1) はジョイスティックの状態を読みとります．どちらかは常に0になるはずですから，両者のビットごとの論理和(OR)

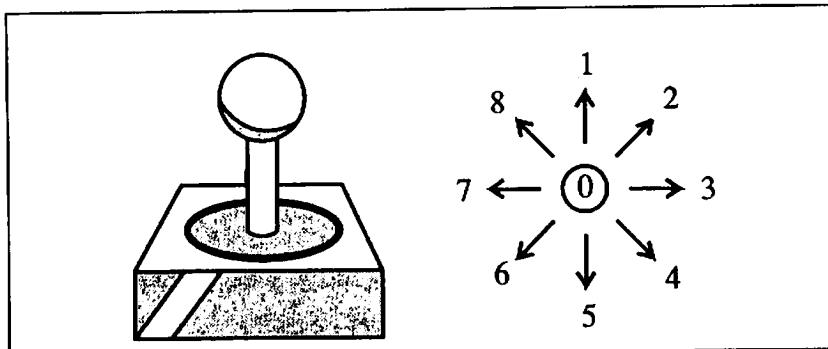



図3.10 ジョイスティックの押されている方向

をとると，「カーソルキーまたはジョイスティック」の状態が得られます．

さて，このようにして得られた移動方向から，配列Xtrns [] とYtrns [] を参照して，x方向，y方向の移動量を決めます．たとえば移動方向が1のときは上方向への移動を意味しているのですから，Xtrns [1] は0，Ytrns [1] は-64 としておくわけです．これらをx, y座標それぞれに加え戦闘機のスプライトを表示すると戦闘機が移動します．

```
 Ø: #include <stdio.h>
 1: #include <msxclib.h>
 2:
 3: char ptsprØ[32] = {
 4:     Øx ØØ, ØxØ3, ØxØ3, Øx13, Øx13, Øx16, Øx14, Øx3d,
 5:     Øx3c, Øx35, Øx75, Øx75, Øx7c, Øx7e, Øx77, Øx33,
 6:     Øx8Ø, ØxcØ, ØxcØ, Øxc8, Øxc8, Øx68, Øx28, Øx3c,
 7:     Øx3c, Øx2c, Øx2e, Øx2e, Øx3e, Øx7e, Øxee, Øxcc };
 8:
 9: int Ytrns[9] = { Ø,-64,-45,  Ø, 45, 64, 45,  Ø,-45 };
1Ø: int Xtrns[9] = { Ø,  Ø, 45, 64, 45,  Ø,-45,-64,-45 };
11:
12: VOID move()
13: {
14:     static int x = 12Ø * 64;     ………… 座標はすべて64倍
15:     static int y = 12Ø * 64;             して、精度をあげる
16:     int i;
```

```
17:
18:     i = gtstck(0) | gtstck(1);    …… カーソルキーまたは
19:                                         ジョイスティック
20:     x += Xtrns[i];
21:     if (x >= 225 * 64)
22:         x = 224 * 64;
23:     else if (x < 16 * 64)
24:         x = 16 * 64;
25:
26:     y += Ytrns[i];
27:     if (y >= 165 * 64)
28:         y = 164 * 64;
29:     else if (y < 28 * 64)
30:         y = 28 * 64;
31:
32:     putspr(0, x/64, y/64, 15, 0);    ….. 表示時に座標
33: }                                         を 1/64 とする
34:
35: VOID main()
36: {
37:     int i;
38:
39:     color(15, 1, 1);
40:     screen(1);
41:     inispr(2);
42:     sprite(0, ptspr0);
43:     while ((snsmat(7) & 0x80) != 0) {
44:         move();
45:         for (i = 0; i < 500; ++i);
46:     }
47:     color(15, 0, 0);
48:     screen(0);
49:     kilbuf();
50: }
```

リスト3.20　GTSTCK.C

# gttrig関数

【機能】 トリガーボタンの状態を調べます

【仕様】 #include　<msxclib.h>　……　ヘッダー・ファイル

TINY gttrig (no)
int no;　……　ジョイスティック番号

【説明】

第1パラメータで指定した装置（ジョイスティックあるいはマウス）のトリガーボタン，またはスペースキーが押されているかどうかを調べます．この関数は，MSX BIOSの00D8hをコールすることで実現しています．

押されているときは1，押されていなかったときは0のTINY型の値が返されます．ジョイスティック番号と装置の対応は以下のとおりです．

0：スペースキー
1：ジョイスティック1のトリガーボタン1，マウス1の左ボタン
2：ジョイスティック1のトリガーボタン2，マウス1の右ボタン
3：ジョイスティック2のトリガーボタン1，マウス2の左ボタン
4：ジョイスティック2のトリガーボタン2，マウス2の右ボタン

【用例】

トリガーボタン，またはスペースキーを押すと戦闘機から弾が発射されます．5発の連射が可能ですが，トリガーを押し続けているとかえって遅くなるように工夫してあります．リターンキーを押すとプログラムは終了します．なお，ジョイスティックを使用する場合はポート1に接続してください．

グローバル配列変数Ypos[]は5発の弾のy座標を表し，この値が255のときは画面上に出現していない未使用の状態を示します．

beam関数は，5つの弾すべての移動処理をおこないます．画面上に出現している弾（y座標の値が255でない弾）は，常に2ドットずつ上昇を繰り返します．弾のスプライトはYpos[]の座標から15を引いた座標に表示し，たとえばy座標が0のとき実際に表示される位置は-15，つまりほんのわずかシッポが見えるという状態になるわけです．

弾の出現処理はshot関数がおこないます．トリガーが押されると，未使用の弾を探し，そのy座標を発射地点の182とします．あまり一度に弾が発射されないように，この関数はwaitという変数で発射間隔を管理しています．弾を発射した直後はwaitの値は36です．そしてトリガーを押していないときはwaitは4ずつ減っていき，トリガーを押していると1ずつ減っていき，これが0になると次の弾が発射できるようになります．これによって，押しっぱなしにしているときよりも，適度な間隔で押し直したほうが効率のよい連射になるという機能が備わります．

```
 0: #include <stdio.h>
 1: #include <msxclib.h>
 2:
 3: char Ypos[5];
 4:
 5: char Ptspr0[32] = {
 6:     0x01, 0x13, 0x13, 0x37, 0x37, 0x76, 0x74, 0x54,
 7:     0xfc, 0xf4, 0xb4, 0xb4, 0xbe, 0xf7, 0xf7, 0x63,
 8:     0x80, 0xc8, 0xc8, 0xec, 0xec, 0x6e, 0x2e, 0xaa,
 9:     0xbf, 0xaf, 0x2d, 0xad, 0x7d, 0xef, 0xef, 0xc6 };
10:
11: char Ptspr1[32] = {
12:     0x01, 0x03, 0x03, 0x03, 0x03, 0x03, 0x01, 0x02,
13:     0x01, 0x02, 0x01, 0x00, 0x01, 0x00, 0x01, 0x00,
14:     0x80, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0, 0xc0,
15:     0x00, 0xc0, 0x00, 0x80, 0x00, 0x80, 0x00, 0x80 };
16:
17: VOID beam()
18: {
19:     int i;
20:
21:     for (i = 0; i <= 4; ++i) {
22:         if (Ypos[i] != 255) {
23:             Ypos[i] -= 2;
24:             if (Ypos[i] < 212)
25:                 putspr(i, 120, Ypos[i]-15, 10, 4);
26:             else {
27:                 Ypos[i] = 255;
28:                 putspr(i, 0, 217, 0, 0);
```

```
29:             }
30:         }
31:     }
32: }
33:
34: VOID shot()
35: {
36:     int i;
37:     static int wait = 0;
38:
39:     if ((gttrig(0) | gttrig(1)) == 0) {  ..... スペースキーも
40:         wait -= 4;                             トリガーボタンも
41:         if (wait < 0)                          押されていない
42:             wait = 0;
43:     } else {
44:         --wait;
45:         if (wait <= 0) {
46:             wait = 36;
47:             for (i = 0; i <= 4; ++i) {
48:                 if (Ypos[i] == 255) {
49:                     Ypos[i] = 182;
50:                     break;
51:                 }
52:             }
53:         }
54:     }
55: }
56:
57: VOID main()
58: {
59:     int i;
60:
61:     color(15, 1, 1);
62:     screen(2);
63:     inispr(2);
64:     sprite(0, Ptspr0);
65:     sprite(4, Ptspr1);
66:     putspr(5, 120, 170, 15, 0);
67:     for (i = 0; i <= 4; ++i)
68:         Ypos[i] = 255;
69:
70:     while ((snsmat(7) & 0x80) != 0) {
71:         shot();
72:         beam();
73:         for (i = 0; i < 150; ++i);
74:     }
75:
76:     screen(0);
77:     kilbuf();
78: }
```

リスト3.21 GTTRIG.C

# gtpad関数

【機能】 マウスの情報を読み取ります

【仕様】 #include　<msxclib.h>

```
int gtpad (no) ;
int no;          ...... 装置ID
```

【説明】

指定したパラメータnoの値により，マウスの各種の状態を調べます．この関数は，MSX BIOSの 00DBhをコールすることで実現しています．したがって，パラメータの値と得られる情報の対応は表 3.7 に示すとおりです．

【用例】

このプログラムはマウスをポート 1 に接続して実行してください．左ボタンを押しながらマウスを動かすと画面に線が引かれます．右ボタンを押すと

| no | 意　味 |
|---|---|
| 12 | ポート 1 のマウスの入力要求をおこなう |
| 13 | ポート 1 のマウスのX座標移動量を得る (−128～+127) |
| 14 | ポート 1 のマウスのY座標移動量を得る (−128～+127) |
| 16 | ポート 2 のマウスの入力要求をおこなう |
| 17 | ポート 2 のマウスのX座標移動量を得る (−128～+127) |
| 18 | ポート 2 のマウスのY座標移動量を得る (−128～+127) |

注：noに 13，14 (あるいは 17，18) を指定して座標値を得る場合は，予めnoに 12 (あるいは 16) を指定して関数を呼び出す必要があります．

表 3.7　gtpad関数で得られる情報

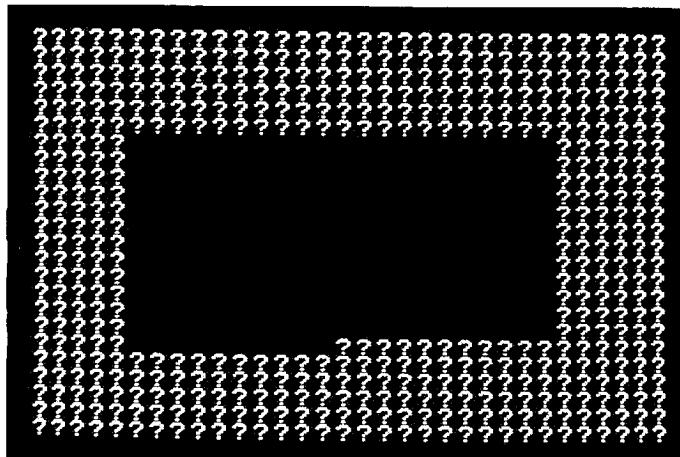
画面をクリアします．リターンキーを押すとプログラムは終了します．

mouse関数はカーソルの移動をおこないます．まずgtpad (12) で入力を要求し，x座標，y座標それぞれの移動量をgtpad (13)，gtpad (14) で読み取ります．これを現在のカーソル位置Cx, Cyにそれぞれ加えれば，新しいカーソル位置が計算できます．

```
 ∅: #include <stdio.h>
 1: #include <msxalib.h>
 2: #include <msxclib.h>
 3:
 4: char Spr[8] = {
 5:     ∅xfc, ∅xf∅, ∅xf8, ∅xfc, ∅xbe, ∅x9f, ∅x∅e, ∅x∅4 };
 6:
 7: char Col[8] = {
 8:     4∅, 4∅, 4∅, 4∅, 4∅, 4∅, 4∅, 4∅ };
 9:
1∅: int Cx = 128;
11: int Cy = 1∅6;
12:
13: VOID mouse()
14: {
15:     gtpad(12);              ……… マウス入力要求
16:     Cx += gtpad(13);        ……… x 座標の読み取り
17:     Cy += gtpad(14);        ……… y 座標の読み取り
18:     if (Cx < 8)    Cx = 8;
19:     if (Cx > 24∅) Cx = 24∅;
2∅:     if (Cy < 8)    Cy = 8;
21:     if (Cy > 2∅∅) Cy = 2∅∅;
22:     putspr(∅, Cx + 1, Cy, ∅, ∅);
23: }
24:
25: VOID draw()
26: {
27:     int x, y;
28:
29:     do {
3∅:         x = Cx;
31:         y = Cy;
```

```
32:             mouse();
33:             if (gttrig(1))    ·············· 左ボタンは線を描く
34:                 line(Cx, Cy, x, y, 15, Ø);
35:             if (gttrig(3))    ·············· 右ボタンは画面クリア
36:                 filvrm(Ø, Øx6aØØ, Ø);
37:         } while ((snsmat(7) & Øx8Ø) != Ø);
38: }
39:
4Ø: VOID main()
41: {
42:     ginit();
43:     color(15, 1, 1);
44:     screen(5);
45:     inispr(Ø);
46:     sprite(Ø, Spr);
47:     colspr(Ø, Col);
48:     draw();
49:     screen(Ø);
5Ø:     kilbuf();
51: }
```

リスト3.22　GTPAD.C

VRAM操作はMSXを扱うための基礎技術

VPOKE.C(p.64) の実行画面

# gicini関数

【機能】 PSGの初期化をおこないます

【仕様】 #include　<msxclib. h>

VOID gicini ()

【説明】

PSGを初期化することができます．このときPSGのレジスタは表3.8のように設定されます．この関数は，MSX BIOSの0090 hをコールすることで実現しています．

【用例】

sound関数の項を参照してください．

| レジスタ | 内容 | 初期値 |
|---|---|---|
| 0 | チャンネル1周波数 | 55 h |
| 1 | | 00 h |
| 2 | チャンネル2周波数 | 00 h |
| 3 | | 00 h |
| 4 | チャンネル3周波数 | 00 h |
| 5 | | 00 h |
| 6 | ノイズ周波数 | 00 h |
| 7 | チャンネル設定 | 68 h |
| 8 | チャンネル1音量 | 00 h |
| 9 | チャンネル2音量 | 00 h |
| 10 | チャンネル3音量 | 00 h |
| 11 | エンベロープ周期 | 0Bh |
| 12 | | 00 h |
| 13 | エンベロープパターン | 00 h |

表3.8　PSGレジスタの初期値

# sound関数

【機能】 効果音を発生させます

【仕様】 #include　<msxclib. h>

VOID sound (rg, dt)
int rg;　　　…… PSGのレジスタ番号
int dt;　　　…… 書き込むデータ

【説明】

MSXの音楽機能としてPSG (Programmable Sound Generator) というLSIが標準装備されています．この関数は，MSX BIOSの0093hをコールして，PSGを直接操作することで音を出しています．

PSGには3チャンネルのトーンジェネレータとボリュームコントロールアンプがあり，それぞれ独立していますので，1つ1つ個別に音程と音量を指定できます．またノイズジェネレータおよびエンベロープジェネレータも備

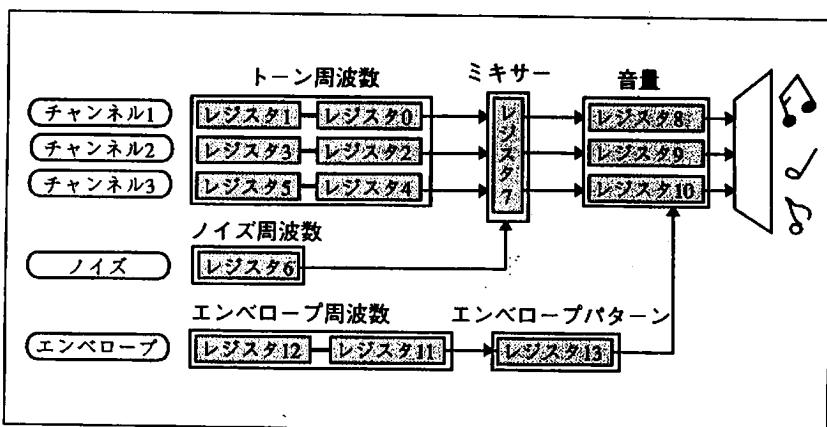



図3.11　PSGのブロックダイヤグラム

えていますが，各々1つしか存在していないために，これは各チャンネルで共用します．図3.11はこのようなPSGの仕組みを表すものです．

PSGは14個のレジスタを持ち(他に2個のレジスタがあるが，音の発生とは無関係なので説明は省略)，そこにデータを書き込むことで動作がコントロールされます．PSGレジスタの構成は図3.12のようになっています．

・レジスタ番号0〜5

チャンネル1，2，3の音のトーン周波数を指定します．トーン周波数は12ビットのデータです．表3.9は標準的な音階を発生させるためのレジスタ設定値です．

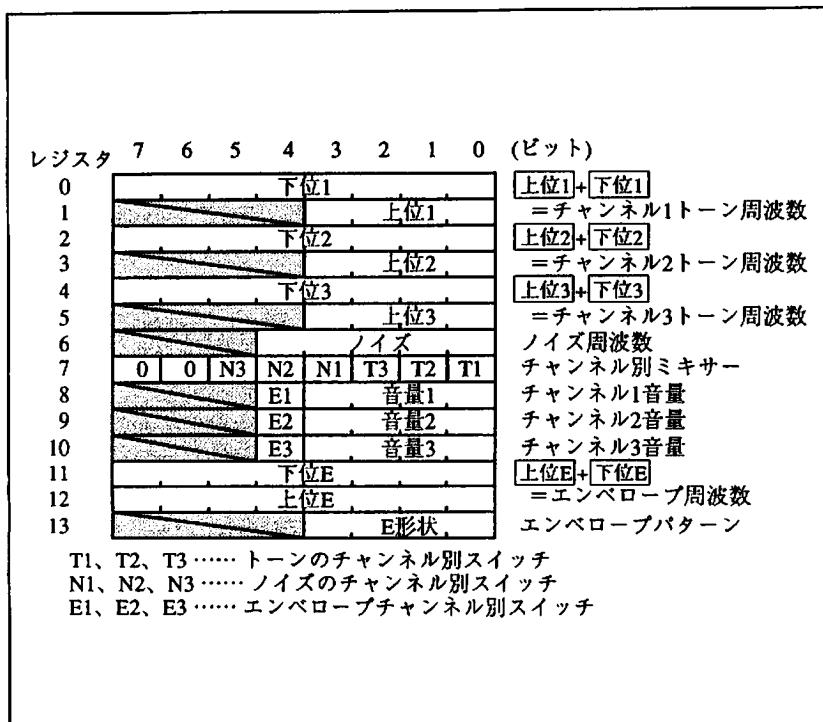



図3.12 PSGレジスタの構成

|  | オクターブ |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
| 音名 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
| C | D5Dh | 6AFh | 357h | 1ACh | D6h | 6Bh | 35h | 1Bh |
| C# | C9Ch | 64Eh | 327h | 194h | CAh | 65h | 32h | 19h |
| D | BE7h | 5F4h | 2FAh | 17Dh | BEh | 5Fh | 30h | 18h |
| D# | B3Ch | 59Eh | 2CFh | 168h | B4h | 5Ah | 2Dh | 16h |
| E | A9Bh | 54Eh | 2A7h | 153h | AAh | 55h | 2Ah | 15h |
| F | A02h | 501h | 281h | 140h | A0h | 50h | 28h | 14h |
| F# | 973h | 4BAh | 25Dh | 12Eh | 97h | 4Ch | 26h | 13h |
| G | 8EBh | 476h | 23Bh | 11Dh | 8Fh | 47h | 24h | 12h |
| G# | 86Bh | 436h | 21Bh | 10Dh | 87h | 43h | 22h | 11h |
| A | 7F2h | 3F9h | 1FDh | FEh | 7Fh | 40h | 20h | 10h |
| A# | 780h | 3C0h | 1E0h | F0h | 78h | 3Ch | 1Eh | Fh |
| B | 714h | 38Ah | 1C5h | E3h | 71h | 39h | 1Ch | Eh |

表3.9　トーン周波数一覧

・レジスタ番号6

ノイズ周波数を指定します．ノイズ周波数は0から31の数値です．

・レジスタ番号7

各チャンネルから音を出すかどうかを指定します．このレジスタは図3.13のようにビットが割り当てられています．そして，各ビットが〝0″のとき，そのビットに対応するトーンあるいはノイズが出力されます．

・レジスタ番号8,9,10

チャンネル1，2，3のそれぞれの音量を指定します．音量は下位4ビットのデータで0から15まで変化させられます．またビット4を〝1″とすることで，音量をエンベロープの波形に合わせて変化させられます．エンベロープを使用するときは音量の指定は無視されます．

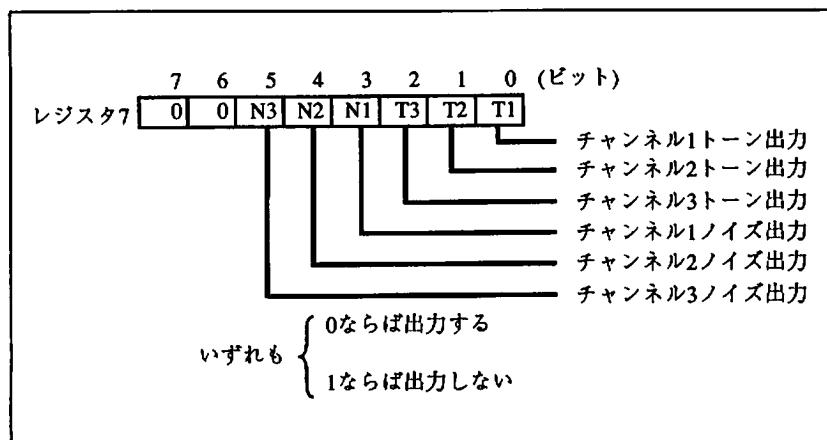



図3.13 各チャンネルの出力選択

| データ | エンベロープパターン | データ | エンベロープパターン |
|---|---|---|---|
| 0~3,9 | | 11 | |
| 4~7,15 | | 12 | |
| 8 | | 13 | |
| 10 | | 14 | |

図3.14 エンベロープの波形の設定

・レジスタ番号 11,12

エンベロープの周波数を指定します. エンベロープ周波数は 16 ビットのデータです.

・レジスタ番号 13

エンベロープパターンを指定します. エンベロープパターンは, 図3.14 のとおりです.

【用例】

ヘリコプターの音を発生させます．何かキーを押すと音は止まります．

チャンネル1から2000 Hzのトーンおよび3800 Hzのノイズを出力し，さらにエンベロープパターン12を周期300 Hzで使用します．このような設定により疑似的にヘリコプタの音を発生させています．

プログラム終了の前にgicini関数を実行していますが，これは音を止めるためのいちばん手軽な方法です．

```
 0: #include <stdio.h>
 1: #include <msxclib.h>
 2:
 3: VOID main()
 4: {
 5:     static int datapsg[14] = {
 6:         55, 0, 0, 0, 0, 0, 29, 54,
 7:         16, 0, 0, 116, 1, 12 };
 8:     int i;
 9:
10:     gicini();     ………… PSGの初期化
11:
12:     for (i = 0; i < 14; ++i)
13:         sound(i, datapsg[i]);   …… 14個のレジスタを設定
14:
15:     getch();
16:     gicini();     ………… 音を消して終了
17: }
```

リスト3.23　SOUND.C

# calbio関数

【機能】 MSXのROM-BIOSの機能を呼び出します

【仕様】 #include　<msxalib. h>

VOID calbio (ad, rg)
unsigned ad;　………… BIOSコール先のアドレス
regs ＊rg;　………… regs構造体へのポインタ

【説明】

calbio関数はBIOSを呼び出す関数です. MSXは, たとえ機種が異なっていても, あるいはハードウェアの変更や機能拡張がおこなわれたとしても, 「BIOS」と呼ばれる入出力に関するサブルーチンを利用する限り, 問題なく作動できるように互換性が保たれています.

calbio関数の第2パラメータは, Z80CPUのレジスタを示すregs型の構造体へのポインタです. このregs型の構造体はf, a, bc, de, hlという名の4つのメンバーで構成され, それぞれ同名のCPUのレジスタに対応しています. 構造体regsはヘッダーファイル"msxalib. h"内で, 以下のように定義されています.

```
struct regs {
    char        f;
    char        a;
    unsigned    bc;
    unsigned    de;
    unsigned    hl;
};
```

このregs構造体に値をセットしてcalbio関数を呼び出すと，各レジスタに値が引き渡されてBIOSが実行されます．また，関数実行後にregs構造体を調べると，BIOS呼び出し終了時点のレジスタの値がわかります．

【用例】

capson関数はCAPSキーのランプを点灯し，capsoff関数はCAPSキーのランプを消します．そして，これらの関数を交互に適当な間隔をおいて呼び出すことで，点滅をさせます．残念ながらCAPSキーのランプが取り付けられていないMSXでは，この動作を確認することはできません．あしからず．

```
 0: #include <stdio.h>
 1: #include <msxalib.h>          ............ struct regs 構造体はこの
 2:                                           ファイルの中で定義されている
 3: VOID capson()
 4: {
 5:     struct regs r;
 6:
 7:     r.a = 0;                   ............ A レジスタに 0 を入れて
 8:     calbio(0x0132, &r);        ............ 0132h 番地の BIOS をコール
 9: }
10:
11: VOID capsoff()
12: {
13:     struct regs r;
14:
15:     r.a = 1;                   ............ A レジスタに 1 を入れて
16:     calbio(0x0132, &r);        ............ 0132h 番地の BIOS をコール
17: }
18:
19: VOID main()
20: {
21:     int i, j;
22:
23:     for (i = 0; i < 10; ++i) {
24:         capson();
25:         for (j = 0; j < 20000; ++j);
26:         capsoff();
27:         for (j = 0; j < 20000; ++j);
28:     }
29: }
```

リスト3.24　CALBIO.C